TI-99/4A GAME PROGRAMS

BY FREDERICK HOLTZ

TI-99/4A GAME PROGRAMS

TI-99/4A GAME PROGRAMS

BY FREDERICK HOLTZ

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION
THIRD PRINTING

Library of Congress Cataloging in Publication Data

Holtz, Frederick.
TI-99/4A game programs.

Includes index.
1. Games—Data processing. 2. TI-99/4A (Computer)
I. Title.
GV1469.2.H64 1983 794.8'2 83-10544
ISBN 0-8306-0730-7
ISBN 0-8306-1630-6 (pbk.)

Cover photography by The Ziegler Photography Studio of Waynesboro, PA.

Contents

Acknowledgments

The author would like to gratefully acknowledge the assistance of Texas Instruments, Inc. and the Texas Instruments Learning Center for their cooperation in researching this book. Without their able assistance, this work would not have been possible.

Introduction

At long last, the microcomputer for *any* home is available in the form of the Texas Instruments TI-99/4A. As of this writing, a $100 manufacturer's rebate is in effect from Texas Instruments, and the basic computer can be purchased for a total price of about $150. When one considers that a machine with the same capabilities only a decade ago might cost in the hundreds of thousands of dollars, it is easy to see just how far microcomputing has come.

The reason I refer to the TI-99/4A as the microcomputer for any home lies in the fact that it is an excellent blend of power and low cost. Certainly, there are many other computers which are far more powerful, but these cost at least twice as much as the TI-99/4A. Unlike some extremely inexpensive computers which offer only marginal power and are severely restricted regarding ease of operation, the TI-99/4A operates like a "real" computer (which it certainly is). It makes an excellent training machine for children and adults alike who may wish to move on to a personal computer in the near future. Also, the TI-99/4A is one of the few home computers that are built around a 16-bit microprocessor. This means the machine should be able to expand its communications capabilities to meet the trends of the computer market.

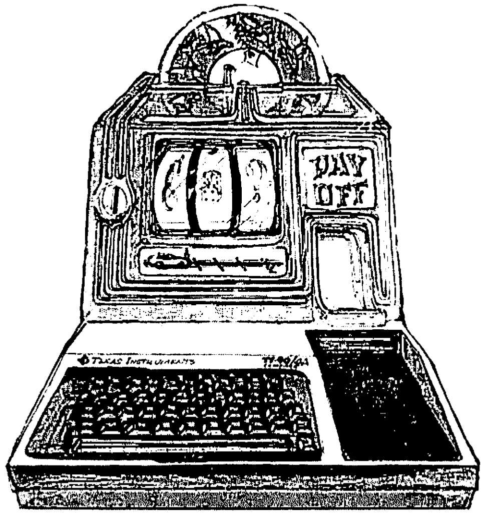
While the TI-99/4A is a true computer, it also may be used in a manner similar to the way video game machines, which are traditionally connected to television receivers, are used. The TI-99/4A contains a ROM cartridge slot in addition to having the capability of reading and writing to cassette and disk. With the cartridge slot, it's a simple matter to plug in a game module and use the machine just like you would a video game.

This book concentrates on the game aspect of this excellent little computer, although the game cartridge slot is never used, since all programs contained in this book are input via the keyboard. No, the game programs contained here are not what you would classify as true video games. Most are text-mode computer exercises that revolve around random chance or involve the skill of the operator. A few games do include on-screen animation, and you will see a fairly large number of on-screen graphics incorporated in the programs.

All programs in this book were written and tested on a TI-99/4A in basic form. This means that you can get by with a minimum configuration, which is the computer itself, the modulator, and a television receiver (black and white or color). More than likely, you will also want to have on hand a cassette recorder and the optional computer/cassette adapter cord, but the latter two are not necessary. The point here is that you do not have to have any options, such as a speech synthesizer, expansion chassis, or RS-232 interface to utilize every game in this book to its maximum. Nor do you need the Extended BASIC firmware package available from Texas Instruments. The latter is most conducive to easily programming different types of video games, but again, this book is aimed at the individual with the basic configuration. If you've purchased your TI-99/4A computer and have a television receiver, you need nothing else to input and enjoy any game in this book.

To Connie and C. L. Longerbeam who I've come to know only recently, but whose friendship seems to have been there forever.

Chapter 1



TI-99/4A Home Computer System

Texas Instruments Incorporated announced an enhanced version of their popular TI-99/4 home computer on May 29, 1981, at the Summer Consumer Electronics Show in Chicago, Illinois. The TI-99/4A was a multi-unit computer system. It contained a 16-bit microprocessor and was one of the first 16-bit home computers to be offered. The new model, announced in Chicago, is designated the TI-99/4A and among other things, includes a new keyboard. The console retains the compact profile, speech capability, and color graphics software capability of the older TI-99/4 but also offers improved functionality. The latter word belongs to Texas Instruments. What it means to the computer user is a lot more versatility. Now you get upper- and lowercase letters, along with numbers, punctuation, and symbols, all arranged on a standard typewriter keyboard. The shift key activates the uppercase characters and can be locked in place by depressing the alpha lock key.

The TI-99/4A (Fig. 1-1) has a built-in automatic repeat function that was not available on the older TI-99/4. This is useful in formatting tabular data and in developing fairly complex graphic designs. By holding the space bar down and depressing any alphabetic or other symbol key, that character is repeated until the key is released.

One of the special keys is the function key (FCTN). When this key is depressed, along with certain designated number keys, you get special computer functions such as delete, insert, erase, clear, begin, proceed, aid, redo, and backspace. The control key (CTRL) is used specifically for communications applications. This includes communicating with an-

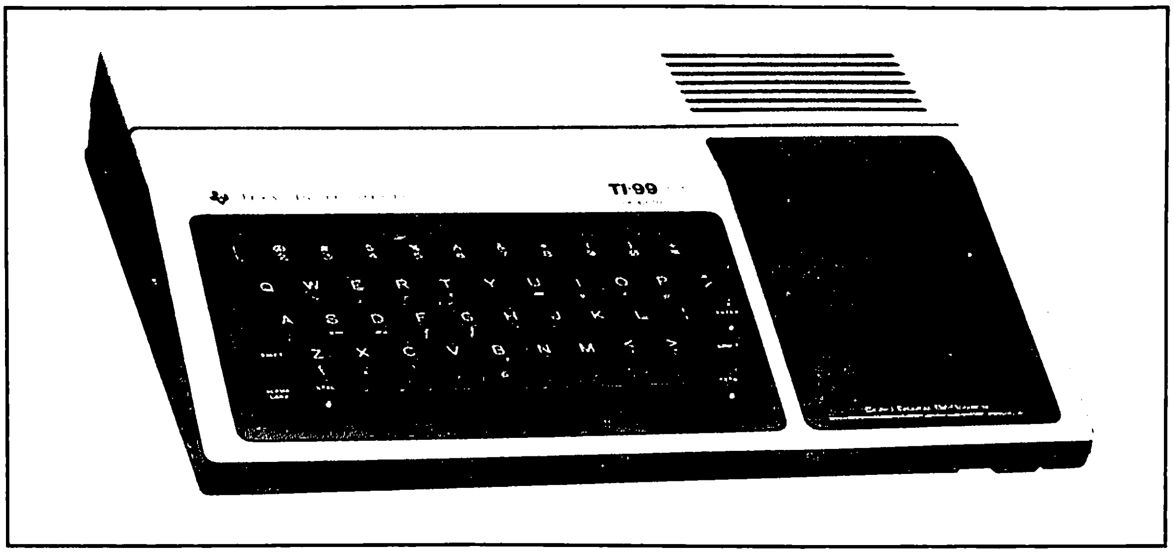


Fig. 1-1. The TI-99/4A microcomputer.

other home computer or even with a remote home information service. A two-level strip overlay is included with the console to help you identify the keys that are used in combination with the FCTN and CTRL keys (Fig. 1-2). For additional identification, control keys and the CTRL key are specified by red symbols; the function keys have gray symbols.

TI BASIC, the standard language on the TI-99/4A, accepts both upper- and lowercase characters, except in a few special instances. When the list command is entered, the screen displays all reserved words, variable names, and subprogram names in capital letters for easy identification. Actually, the lowercase letters are smaller reproductions of the uppercase character set. A lowercase R is not formed differently from an uppercase R; it is just physically smaller.

The TI-99/4A, like its predecessor, is equipped with a module slot so that solid state command modules may be inserted. These modules may contain Extended BASIC, a more powerful version of TI BASIC, or any of a hundred or so different programs. This computer will also store and read programs on and from cassette tape with the optional cassette interface cable. A disk drive system is available as well for those users who need the additional speed and convenience disks provide. In most instances, however, users will probably stick to the less expensive cassette tape storage medium.

The software for this computer is extensive and allows individuals with no prior computing experience to begin enjoying the machine. By adding peripherals such as disk drives, printers, or telephone modems, the TI-99/4A becomes a quite powerful problem solver for advanced users. It is the only inexpensive home computer that can be programmed to include 16 colors, numerous sound effects, and five musical octaves (with three-part harmony).

Another option is the 10-inch color monitor that accepts the composite video signal directly from the TI-99/4A. This is an option that many users won't buy because a modulator is supplied to connect the console with a television receiver. The picture quality is better with the 10-inch monitor, but the display on a standard color television screen is quite good. The fact that the color monitor option costs more than the basic computer itself is a big factor in selecting which type of monitor to use. If you're going to be heavily involved in color graphics work with the TI-99/4A, you will certainly want to consider the excellent reproduction provided by the small, 10-inch monitor.

Resident memory in the TI-99/4A is specified as 72K bytes. This can be a little misleading since most companies rate their random-access memory only. The TI-99/4A includes 16K bytes of data storage memory. This is also known as random-access memory, read-write memory, or simply RAM. This is generally the minimum amount of RAM recommended for any type of computer system, although very few single programs will use even half of the available memory. With 16K RAM, any program you run cannot require more than 16K of memory.

This limitation doesn't mean that once you've written one program that consumes about 16K, your machine is useless until more memory is added. Two different programs cannot be run simultaneously on this microcomputer. When you write a program, you may want to store it in permanent memory. This is usually a cassette tape for the TI-99/4A, but it may also be a magnetic disk. Once the program information is transferred from RAM to disk or cassette, you are free to erase the program from RAM and begin a new one. When this one's completed, you can store it on cassette or disk as well. Suppose you want to run one of the stored programs. You simply load the information from the storage medium into RAM.

I find that the majority of individuals who buy a microcomputer actually buy more memory than they will ever use. It takes a long program to fill 16K RAM, so I suggest you try the minimum configuration to determine what your memory requirements are.

In addition to the 16K RAM, the TI-99/4A contains 26K bytes of read-only memory (ROM). These chips contain the instructions needed for your computer to know it's a computer. Additionally, they include information to allow the microprocessor to accept information from the keyboard and perform all of the functions that allow you to enter a program in TI BASIC.

The 26K bytes of ROM and 16K bytes of RAM, a grand total of 42K bytes, is all the memory that comes built *into* the basic TI-

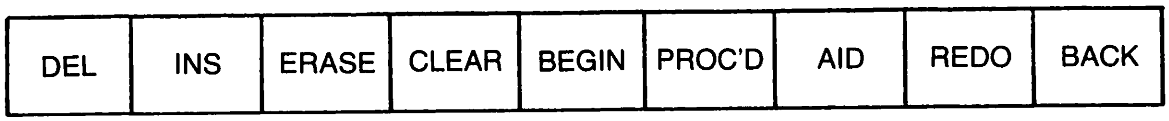


Fig. 1-2. A two-level strip overlay fits above the top row of keys to identify special function keys.

99/4A computer. When Texas Instruments states that memory resident in the basic system is 72K bytes, they are including in that figure 30K bytes of ROM that is actually contained in an *accessory plug-in* memory module. This extra ROM may contain an optional program or new language. It accomplishes in ROM what might normally be accomplished in RAM through software.

Those of you who require more than 16K of RAM will be happy to know that RAM data storage is expandable to a total of 48K bytes through the memory expansion unit. This option adds 32K bytes to the resident 16K bytes of RAM.

HOME VERSUS PERSONAL COMPUTERS

The TI-99/4A is a home, rather than a personal, microcomputer. The home computer is often called a low-level microcomputer, and the personal computer is often called a high-level microcomputer. The differences between the two groups are becoming less distinct as home computer design and capabilities are being constantly upgraded. The 16-bit microprocessor in the TI home computer, for instance, is far more advanced than the 8-bit microprocessors used in many personal computers.

In general you will find fewer options (such as communications interfaces, disk drives, and mass storage systems) available for home computers than for personal computers, or if available, they will cost more than the cost of the computer itself. Home computers generally have fewer operating features than personal computers. Here too, the differences are disappearing. On the TI, for instance, it is easy to edit, or make program changes in a line without having to retype the entire line. This is a feature that was not available on most personal computers only a few years ago.

The character set and screen display on a personal computer will generally be of higher quality than found on home computers. A true personal computer can also more easily and more quickly display charts and graphs through direct programming methods using the features built into the machine. Because of this capability personal computers may also be referred to as small business computers.

Although many home computers may accomplish the same results, they most often require special software or hardware packages to do so and will do so much slower.

Screen displays are also handled differently. On most personal computers the first line of text appears at the top left-hand corner of the screen, and no scrolling (the automatic movement of displayed lines upward) occurs until after the screen is full. On the TI microcomputer, as on most home computers, the first line of text appears at the bottom left-hand corner of the screen. Upward scrolling begins with the very next line that is typed. This is the easiest way to accomplish on-screen display because it requires less ROM. For most home computer uses this is really no drawback, but it can prove troublesome in more sophisticated personal or business applications.

Another difference in screen display is screen format. Most home computers display characters in only one format, and it is usually about 30 columns wide by 25 rows deep; that is approximately 30 letters can be printed in a single row and you can get up to 25 rows on the

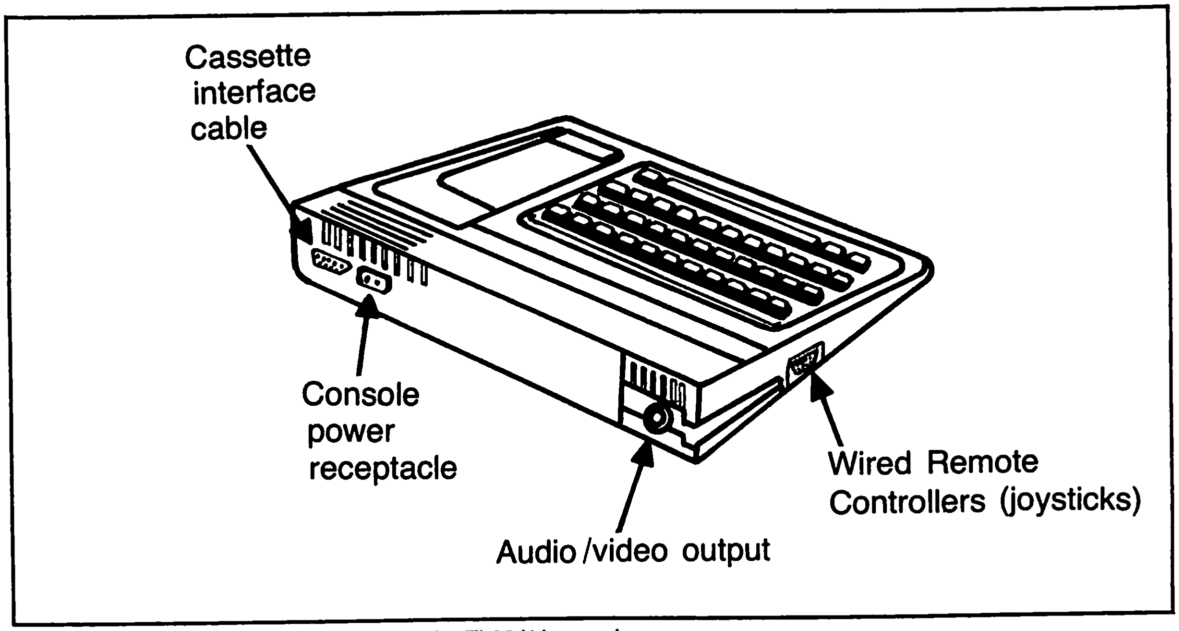


Fig. 1-3. The location of various outlets on the TI-99/4A console.

full screen. The TI-99/4A allows a 32-character wide format with 24 rows on the full screen. True personal computers usually display text in at least two different screen formats: one format approximately 80 columns wide and 24 or 25 rows deep (called high-resolution text mode) and one format 40 columns wide and 25 rows deep (called medium-resolution text format). This type of screen formatting can be a great aid in clearly displaying information without having to do a lot of hunting through screen garbage. Although most home computers display information quite accurately, you may have to look for a second or so longer to distinguish it from other on-screen information.

Because personal computers generally offer more keyboard functions than home computers they make inputting information easier. For example, on the TI-99/4A home computer, you must press the FCTN key simultaneously with another key to move the cursor to the left or right. On most personal computers, cursor movement is controlled by a separate key panel that requires only one finger to operate.

While the extra features of a true personal computer may be very nice, you pay through the nose for them. Keep in mind that the TI-99/4A is available from most discount stores for less than $200. In comparison, the personal computer I have sells for about $2,500. With options such as a disk drive, special monitor, and memory expansion package, it costs well over $6,000. In other words, my personal computer cost thirty times what the TI-99/4A home computer costs.

The question every potential computer owner must ask is, "Do I need everything the expensive machine offers?"

The TI-99/4A is a home computer designed to be used in the home by the average homeowner and home dweller. Texas Instruments states that adults and children with little or no knowledge of computers can easily use the TI-99/4A. Texas Instruments has attempted to achieve (and in my opinion, has) a simple machine built around a high-powered microprocessor. Simplicity is the key here, ergo the standard typewriter keyboard and excellent software packages that are offered.

As an experienced computer operator, I would love to see a separate numeric keypad on the TI-99/4A, just like the one on my personal computer. However, such a keypad is certainly not mandatory, nor even desirable for the beginning or casual home computer operator. It would certainly add to the cost and the complexity of operating the machine.

For the money, the TI-99/4A is one of the best computer buys on today's market. I have tried many different home computers, and I especially like this one because of its standard typewriter keyboard. Many home computers do not offer this feature. The main reason I like the TI though is because its language and keyboard operation are identical or similar to those contained in personal computers. Some home computers seem to program in a completely different manner from most personal computers. They don't, therefore, make very good training aids for children who may someday hope to convert to serious personal computing. If you learn to efficiently program the TI-99/4A, you will be able to easily convert to a full-scale personal computer when the time comes. If you have programmed for quite some time on a personal computer, you will be able to easily convert to the TI-99/4A. There are marked differences in the two types of machines, but there are enough similarities to make converting to either an easy task.

THE CONSOLE

The TI-99/4A console is shown in Fig. 1-4. This is the master unit and contains the microprocessor, the keyboard, solid state software command module input slot, and the video/audio interface.

The TI-99/4A console or master unit is designed around the TMS9900 16-bit microprocessor. Its architecture is 16-bit with 16 general registers. It can address up to 54K bytes of memory and contains four interrupt lines.

Also found in the console is a video display processor chip. It controls display memory and generates the composite video signal used to drive a composite monitor or a video modulator when a color television is used. It displays 24 lines of 32 characters in an 8-by-8 dot matrix. The processor provides 16 colors and for flexibility in the use of color, divides the characters into sets of 8 characters each, with different foreground/background colors. The video display processor addresses up to 16K bytes of RAM for the central processor or display purposes.

There is a third chip in the console unit, the sound controller chip. This chip offers three voices with a 5-octave musical range. It also contains a 15-bit programmable noise source and offers a 100-milliwatt audio output with 30 dB control in increments of 2 dB.

The keyboard is contained within the console. It's known as a 48-key Staggered Qwerty, full travel type. It is very similar to some

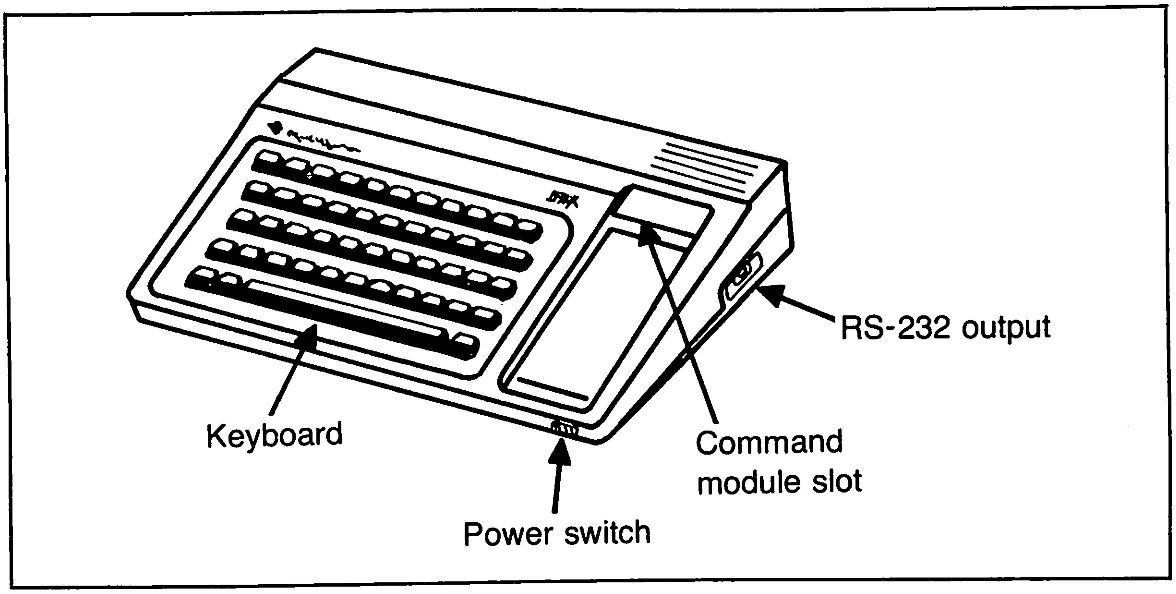


Fig. 1-4. Front view of console.

typewriter keyboards, although you will note a few differences in the placement of certain keys. The number and letter keys, however, are in the usual location.

The console unit itself also contains the 14K byte BASIC Interpreter, along with a graphics language interpreter.

The console is powered from a standard 110-Vac source. The power supply is located in the power cord. There are two arrangements here. One model may locate the transformer a distance from the plug, while another type has the plug-in type of transformer arrangement. Here, the plug prongs exit the transformer body and the entire unit rests against the wall receptacle. It doesn't matter which type you get because both are electrically equivalent. The console draws a maximum of about 20 watts.

The removable power cord attaches at the rear of the unit by means of a 4-pin plug. The location of the various outlets on the console is shown in Fig. 1-3. At the far left, the cassette interface cable is connected to a 9-pin D outlet. Immediately to the right of this receptacle is the console power receptacle, and to the far right is the 5-pin connector for audio/video output from the unit. This connector accepts the cabling from a composite video monitor or from a video modulator when a television receiver is to be used. To one side of the console is another receptacle to accept the joysticks, or as Texas Instruments calls them, the wired remote controllers. This receptacle is identical to that found on the rear of the unit for interfacing with a cassette tape recorder. Do not get these two receptacles mixed up.

Figure 1-4 shows the front of the console. The power switch is located at the lower front right near the command module software slot.

All of the module software is inserted in this slot. To the right of this slot is an output jack for optional peripheral accessories, such as the RS-232 interface. Since a good bit of the microprocessor circuitry is located just beneath the module software slot, it is normal for there to be a bit of heat in this area. The plastic casing in this area will become warm, but not so hot that it's uncomfortable to touch.

THE KEYBOARD

One big advantage of the TI-99/4A over the previous TI-99/4 is the typewriter keyboard (Fig. 1-5). For the most part, the keyboard looks and operates like a standard typewriter keyboard. When you press any key, its lowercase character appears on the screen unless you hold down the shift key at the left or right. When this is done and another key is simultaneously pressed, the uppercase character for that key appears on the screen. There is also an alpha lock key at the bottom left of the keyboard. When this is depressed, the keyboard is locked into uppercase mode. The alpha lock key does not affect the number and punctuation keys. Pressing the alpha lock key one more time will "unlock" the uppercase mode, and return the keyboard to normal lowercase operation. Except for the alphabetical keys, each key's uppercase character is printed at the top of the key face. The lowercase character is printed at the bottom. This is not done for the alphabetical keys because the characters are formed in exactly the same manner for both upper- and lowercase. The only difference between uppercase and lowercase letters will be their size. Some of keys have special functions that are accessed by depressing the FCTN key simultaneously with another key. Some characters formed with the

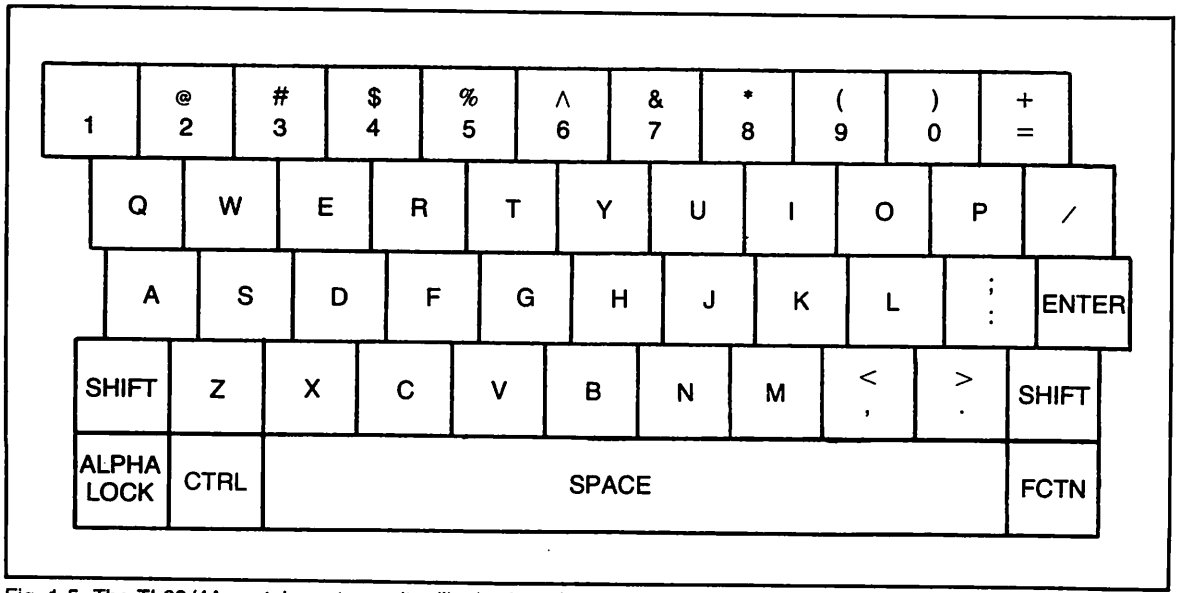


Fig. 1-5. The TI-99/4A contains a typewriter-like keyboard.

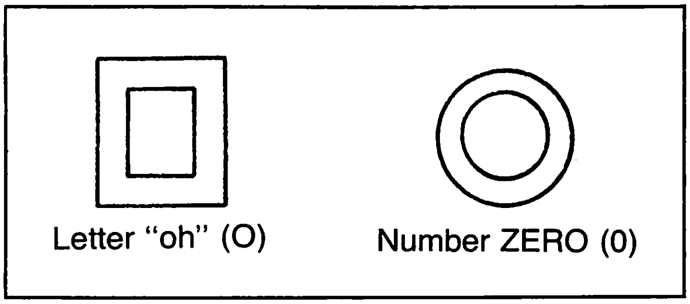


Fig. 1-6. Display of the letter O and the number zero on the TI-99/4A.

aid of the FCTN key are printed on the front or side of the corresponding keys, rather than on the top, as is normally the case with most keyboards.

The number keys on the TI-99/4 console are on the top row. Unlike on some typewriters, you cannot type the lowercase letter "elle" (l) as a substitute for the number one (1), and you can't interchange a zero (0) and the uppercase letter "oh" (O). The computer screen displays the letter O with square corners and the number zero with rounded corners (Fig. 1-6) to make it easier for you to distinguish between them. By pressing the shift key and number keys, symbols become available.

The keyboard has most of the punctuation and symbol keys of a standard typewriter. There are also a few special ones that have particular applications in computer programming and are not found on most typewriters. These punctuation and symbol keys follow the same upper- and lowercase format as other keys: each has two symbols printed on its face. To print the top symbol, you must use the shift key. To print the bottom symbol (lowercase), simply strike the key. Some punctuation marks (quotation marks, for instance) appear on the front of the key. The only way you can type quotation marks or any other symbols on the fronts of keys is by holding down the FCTN key at the lower right of the keyboard while pressing the appropriate key. The special function keys with arrows on them do not print anything on the screen. These are used to move the cursor during the line editing process.

There are other special function keys as well. These are really the number keys that are pressed simultaneously with FCTN. The following is a rundown of the special FCTN key combinations.

When you press FCTN and the key with the arrow pointing toward the left, the cursor moves to the left. The cursor does not erase or change any characters on the screen, but in edit mode it allows you to insert or delete a character by means of other commands. The right arrow key moves the cursor to the right when pressed simultaneously with FCTN. The other two arrow keys, one pointing up and the other pointing down, have different functions according to the application they are being used for and the software itself. When you enter a program, pressing FCTN and up arrow key will cause the lines on the screen to scroll upward.

A special overlay is provided to owners of the TI-99/4A. This metallic strip is fitted over the top of the keyboard to indicate which number key performs which function when pressed simultaneously with the FCTN key. The key bearing the number 1 is labeled as DEL by the overlay. When this key is pressed simultaneously with the FCTN key, a character is deleted from the screen. You usually use this while in the edit mode, or before a program line is entered. Using the cursor positioning keys (right and left arrow keys) discussed earlier, you place the cursor beneath an improper character and then press FCTN and the number 1 key to delete it. When the character is deleted, all other characters move one space to the left to fill in the empty space. By continuing to hold down FCTN and 1, more letters or characters are deleted. All letters to the right of the cursor move toward the cursor.

When you press FCTN and the number 2 key, the insert mode (INS) is accessed and letters or characters may be added to a program line. Let's assume that instead of typing PRINT X, you typed PRIT X. To correct this you enter the edit mode by typing EDIT and the line number that contains the error. Using FCTN and the cursor positioning keys, you move the cursor until it rests beneath the T. Now press FCTN and the number 2 key, and you are in insert mode. Type the letter N, and the N is inserted just before the T. Pressing the enter key saves this line as edited. The delete and insert functions on the TI-99/4A can save a great deal of time by letting you quickly and easily correct mistakes.

When you press FCTN and the number 3 key, the entire program line you are presently typing is erased. This must be done before you press the enter key to commit the line to memory. This is handy in situations where you might be entering a program that is printed in a book. Midway through the entering of a line, you discover that you skipped a line and this one is completely wrong. Press FCTN and the number 3 key, and it's gone. Then begin typing in the correct line.

When you press FCTN and the number 4 key, all execution stops. The key is also used to clear any information from the screen before you press the enter key. Its first feature (execution halt) is most important. On other computers, this might be called a break key or a halt key. To stop a program in the midst of execution, press FCTN and the number 4 key simultaneously. You can now enter other commands in direct mode.

The other numerical keys have special functions in software applications and are labeled by the same overlay strip. The overlay strip is laid out in two levels. The top level of functions is identified by a red dot. These keys are called control keys. These functions are accessed by holding down the CTRL key and the numeric key simultaneously. The second level of functions is identified by a light gray dot. These are accessed by pressing that key while holding down the FCTN key.

The TI-99/4A has several math keys used to insert symbols to indicate math functions. The plus (+), minus (−), and slash (/) indicate addition, subtraction, and division. The asterisk key (*) indicates multiplication, the equal sign key (=) means equal. There is also a caret key (∧) used to indicate the raising of a number to a certain power. For instance, a 5∧2 indicates 5 raised to the second power, or squared. It is necessary to use the shift key to obtain this character, which is found on the number 6 key in uppercase mode. Two other mathematical symbols are found on the keys at the lower right of the keyboard. In lowercase mode, these keys type the comma (,) and the period (.). In uppercase mode the comma key becomes the less than symbol (<), and the period key is the greater than symbol (>).

Another feature of this keyboard is the automatic repeat. If you hold down the space bar or any character key for more than about one second, it goes into repeat mode. To type a series of 5 spaces, press the space bar once and hold it down until your 5 spaces have been printed. The same applies to any character key. This comes in handy in certain graphics applications, where it may be necessary to print a series of 16 Fs, for instance.

The space bar may be used to delete or erase characters from a program line before the line has been committed to memory by pressing the enter key. If you want to erase an entire word, you simply position the cursor at the beginning of the word and hold the space bar down until all letters in the word have been replaced by spaces. (Of course, you can use the FCTN key and delete key for the same basic purpose.)

The feel of the keyboard is quite important to typists who depend on "feel" to put them into a typing rhythm. I would not call the TI-99/4A keyboard crisp, but rather pleasingly spongy and quiet. It does not have the feel of any keyboard I have ever used before. This doesn't mean that it's bad, just different. It's a quiet keyboard; you don't hear the various clicks present with other types of computer keyboards. After ten or fifteen minutes of practice, one becomes adjusted to the keying action, and good typists can fly along at a comfortably rapid pace. I rate the keyboard as excellent for an inexpensive home computer.

ACCESSORIES

A variety of accessories are available for the TI-99/4A computer.

Video Modulator

One item listed as an accessory for the older TI-99/4 is now standard with the TI-99/4A. This is the video modulator (Fig. 1-7), which plugs directly into the console and attaches to the 300-ohm antenna terminals of your color television receiver. The TI900 video modulator is also called by several other names, such as Sup'R Mod, depending on the company from whom you buy it. This is a high-quality Korean-made modulator bought in bulk by many companies and sold under different names. Texas Instruments made a good choice with this modulator, as it's probably one of the most popular types for microcomputer users.

This is an audio and video modulator, so it transmits video information and audio information on the same carrier. The circuitry of your television receiver separates the sound and picture information just as it does with the transmissions from a television station. The picture information is displayed on the picture tube, and the sound is emitted from the internal television speaker.

The Texas Instruments modulator is switch-selectable between channels 3 and 4. Connect the unit to the back of your television set by means of the short length of 300-ohm cable that exits the top. There is also a terminal strip on the side of the modulator, to which

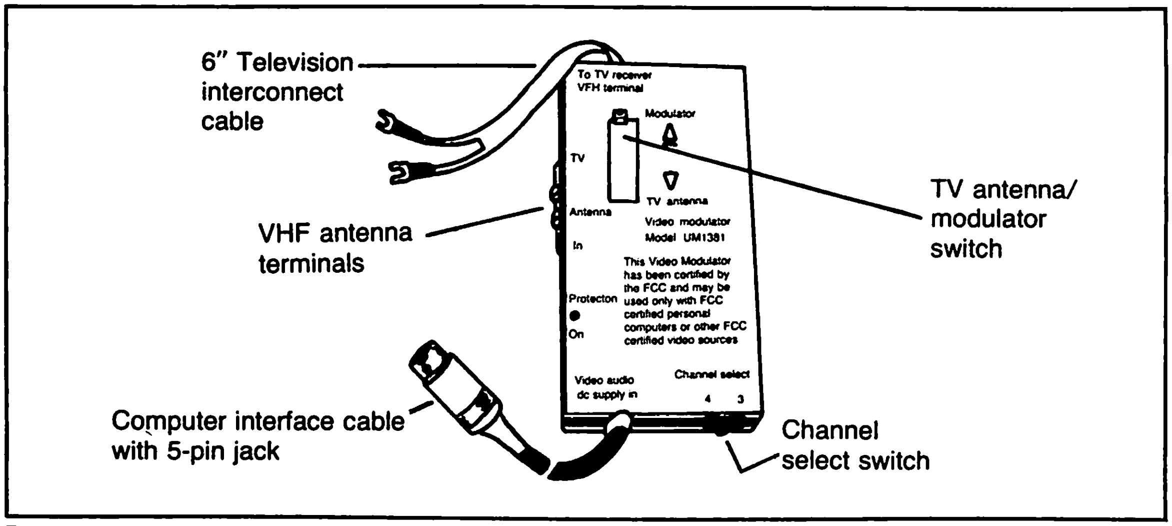

6" Television interconnect cable
VHF antenna terminals
Computer interface cable with 5-pin jack
TV antenna/ modulator switch
Channel select switch
To TV receiver VFH terminal
Modulator
TV
Antenna
In
TV antenna
Video modulator Model UM1381
This Video Modulator has been certified by the FCC and may be used only with FCC certified personal computers or other FCC certified video sources
Protecton
On
Video audio dc supply in
Channel select
4 3

Fig. 1-7. TI900 video modulator.

you can connect your television antenna leads. A switch at the center of the modulator lets you select either the computer or the television antenna for input to the television set. When you want to operate the computer into the television, set this switch in the "computer" position. The "television" position allows for normal television viewing.

The channel selector switch determining the output frequency of the modulator is found at the bottom front. In the left position, the output is on television channel 4; in the right position, your computer output is seen on channel 3. If you have a strong local station on either of these channels, select the other one or for that matter, whichever setting gives you the clearest screen.

Don't be surprised if you hear a few clicks, pops, and other sounds from your television speaker. This is common and can be corrected by turning down the volume. This assumes that you are running a program that is not using the TI-99's sound production capabilities. When using the computer to produce music, leave the television volume up, because the music comes from this speaker.

Another good feature of the modulator supplied by Texas Instruments is its built-in protection. If the computer overdrives the modulator (supplies too much video signal), the protective circuit temporarily disables the device. When the protection circuit is activated, a red light-emitting diode (LED) on the front panel is triggered. You can reset the modulator by turning the mode select switch on its front face to "television" and then back to "computer" again. If the light continues to be triggered, this could be an indication of a defect in the computer or even the modulator. For best results, try to place the computer console at least 3 feet away from the television receiver. This can avoid unwanted video and audio interference from the interaction of the two.

Color Monitor

Shown in Fig. 1-8, the Model PHA 4100 is a high quality color monitor specially matched for use with the TI home computer. The display format is 24 lines of 32 characters per line and the monitor provides excellent color resolution and picture quality. The 10-inch screen has a 192 by 256 dot density ratio and connects to the TI-99/4A computer via a special cable. This eliminates the chance for interference and distortion that can occur in the tuner of a standard television. This is a true color monitor and accepts the composite video directly. Therefore, the PHA 4100 has no tuning. The picture quality using a composite color video monitor is almost always superior to pictures from even the best color television receiver.

The monitor accepts the National Television Systems Committee (NTSC) composite

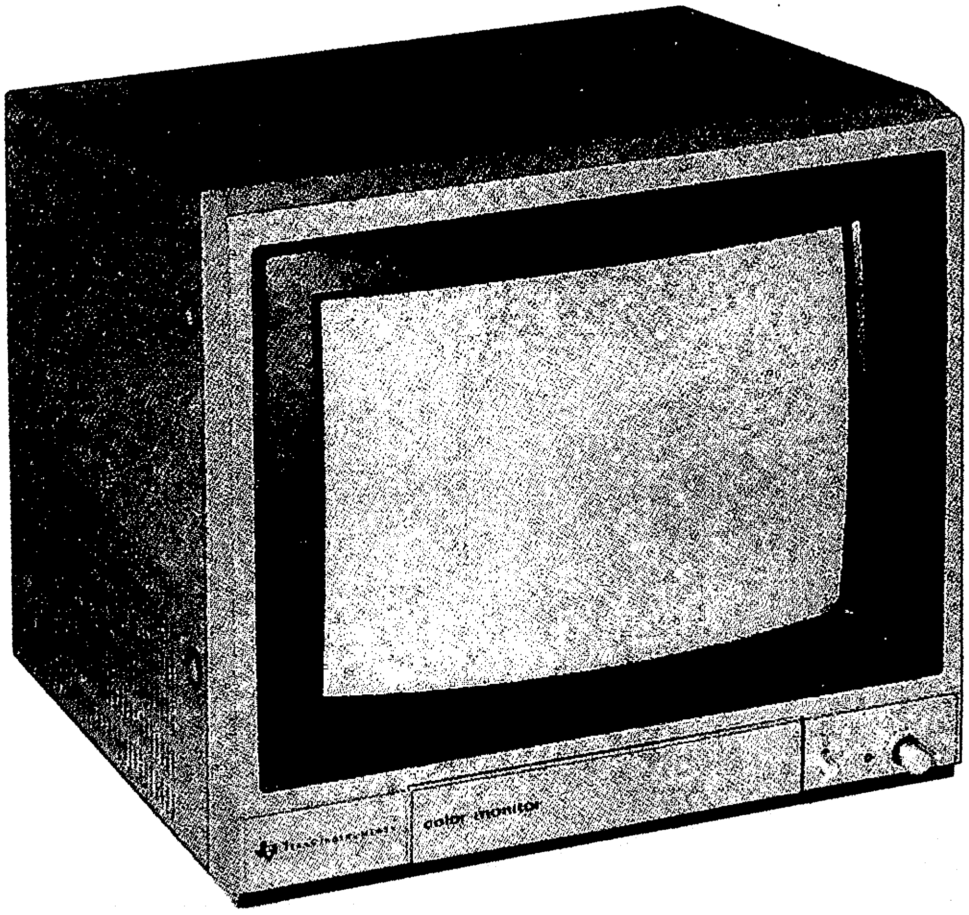


Fig. 1-8. The TI 10-inch color monitor.

video signal at a nominal 1-volt peak-to-peak value. Audio input is delivered at 1 to 2 volts peak-to-peak. The operating, or scan, frequency of this monitor is 15.750 kHz. In addition to the standard on/off and volume controls, you will also find controls for sharpness, tint, color level, contrast, brightness, height, vertical hold, and horizontal hold. The monitor operates from the standard 110-volt household line, consumes about 65 watts, and weighs about 22 pounds.

Texas Instruments warrants all components of the color monitor with the exception of the picture tube for a period of three months from the date of purchase. The picture tube is warranted for a period of two years from the date of purchase.

Thermal Printer

Shown in Fig. 1-9, the solid state thermal printer gives printed copy of any program and/or data run on the TI-99/4A. The printer can also be used with some software applications to print screen displays or generate lists and reports.

The printer can print up to 32 characters per line. It contains its own resident character set, but it can also set special characters defined in software. Other special features included with this device allow you to control the amount of paper that is ejected and the spacing between lines. In many computer applications, a hard copy printout is quite desirable and often necessary. The TI solid state thermal printer is excellent for this purpose. In addition to standard letters and numbers, the printer has 32 predefined graphic symbols for printed charts and graphs.

Printing is done on 3.5-inch thermally sensitive paper, the same type of paper used

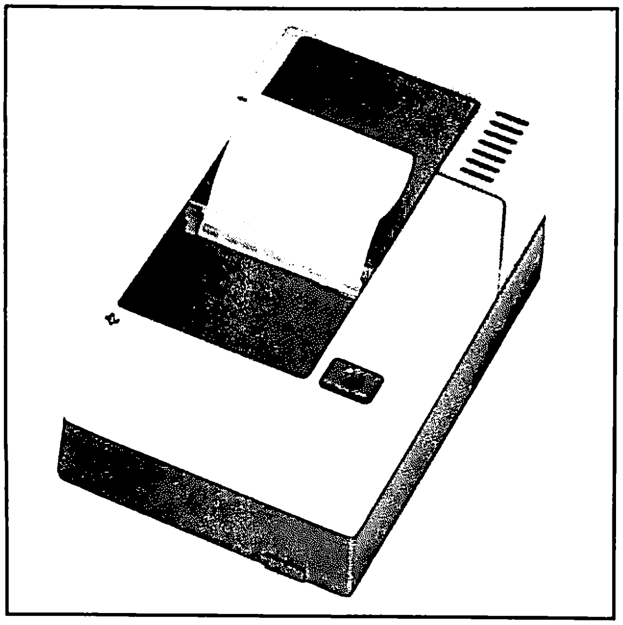


Fig. 1-9. The solid state thermal printer from Texas Instruments.

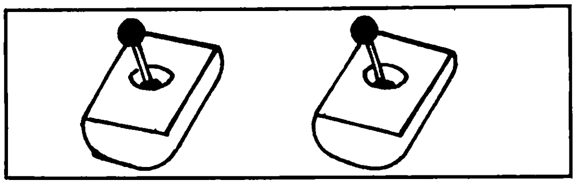


Fig. 1-10. The wired remote controllers from Texas Instruments.

for some of TI's printing commercial calculators. The printer is quite tiny and measures approximately 10 inches by 7 inches by 5½ inches. Because thermal printers normally use fewer moving parts than impact-type printers, they can be far more reliable.

Several software programmable functions are available with this printer. When .U is listed in a program, the printer accepts user-defined characters. If .U is not listed, the printer uses its resident character set. If .S is listed in a program, the printer does not leave any space between printed lines. If it is not listed, the printer leaves a space that is equivalent to the height of 3 rows of dots between the printed lines. When .E is listed in a program, the printer does not eject paper as the program runs. If it is not listed, the printer automatically ejects five lines of blank paper for each open and close statement for the printer. Five lines of blank paper are also ejected before and after each list statement.

The printer is controlled from certain TI command modules and from TI BASIC. The open, print, and close statements in a program control and output data to the printer to produce printed copy when the program is run. The list command tells the computer to print a copy of the program currently in memory.

The TI solid state printer prints approximately 30 characters per second and offers upper- and lowercase characters. It must be used with thermal printing paper (PHA-1950), available only from Texas Instruments. Other thermal papers may damage the printer and void the warranty, which is in effect for 90 days from the date of purchase.

Wired Remote Controllers

Shown in Fig. 1-10, these are often called joysticks and allow you greater freedom and versatility in the controlling of graphics, games, and sound on your computer. Without the joysticks, it is necessary to press one or more keys to effect similar control, which may not be as precise as that offered by the remote controller. The remote controllers are required for certain software offered by Texas Instruments and are a must for programmers who wish to concentrate on developing complex computer games.

RS-232 Interface

The Texas Instruments RS-232 interface (Fig. 1-11) is a communications adapter that lets you connect serially formatted devices, including those from other manufacturers, to the TI-99/4A. It is not required for the use of TI-99/4A peripherals manufactured by Texas Instruments (with the exception of the telephone coupler). With the RS-232 interface, you can list programs on a printer, send and receive data from a terminal, exchange TI BASIC programs directly between TI home computers, etc.

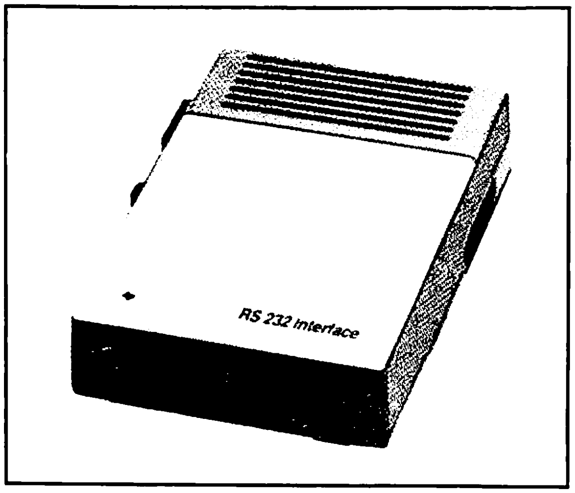


Fig. 1-11. The RS-232 interface allows communications with serially formatted devices.

With the addition of the telephone coupler (modem) or other standard modem or acoustic coupler and the RS-232 interface, the TI-99/4A can talk with other computers and terminals over standard telephone lines. You can access an office computer or time-sharing network using the TI-99/4A as a remote terminal to send and receive data. This two-way communication permits interactive programming and distributed processing functions to be performed between two or more TI-99/4A computers or by using the TI-99/4A as a remote terminal for another computer system.

The RS-232 interface is programmable so you can exchange data with a variety of serially formatted devices. Using TI BASIC, you can select baud rate, the number of bits, parity, and the number of stop bits. This lets you interface with low- and high-speed peripherals including printers, plotters, video display terminals, and other computers.

The interface is capable of outputting information at rates 110, 300, 600, 1200, 2400, 4800, or 9600 bits per second.

Several software programmable functions are available and include:

☐ *Carriage Return.* Automatically added to the end of all output records unless disabled. If disabled, forces nulls and linefeed to be disabled also.

☐ *Nulls.* Normally disabled, but if enabled, will automatically add 6 null characters between the carriage return and the linefeed characters.

☐ *Linefeed.* Automatically added after carriage return character unless disabled.

☐ *Echo.* Automatically echoes all received data on a particular port back to the device connected to that port. Also enables the remote terminal device to edit the data record before the console receives it.

☐ *Parity.* Normally disabled, but if enabled, will check for parity errors and generate an error code if any are found.

The RS-232 interface also contains all the software necessary to interface with the TI Home Computer File Management System and is controlled from TI BASIC. The open, close, input, print, old, and save statements can be used to input and output data through the two ports of the RS-232 interface. The input and print statements can input and output data to a terminal. The old and save commands can transfer a copy of a TI BASIC program from one TI home computer to another.

Two serial ports are provided by this device, and connection is by means of cables using EIA RS-232-C standard 25-pin male connectors. Seven signals are used:

SERIAL DATA IN
SERIAL DATA OUT
CLEAR TO SEND
DATA SET READY
DATA CARRIER DETECT
DATA TERMINAL READY
SIGNAL GROUND

This device is operated from the ac line (115 volts) and consumes a maximum of 20 watts of power during normal operation.

Peripheral Expansion System

The TI peripheral expansion system (Fig. 1-12) lets you add accessories to your computer in a single unit by inserting them in the slots provided. The package includes the expansion system and the peripheral expansion card with a connecting cable. The latter pair combine to serve as an interface between the computer console and the accessories mounted in the unit. With the peripheral ex-

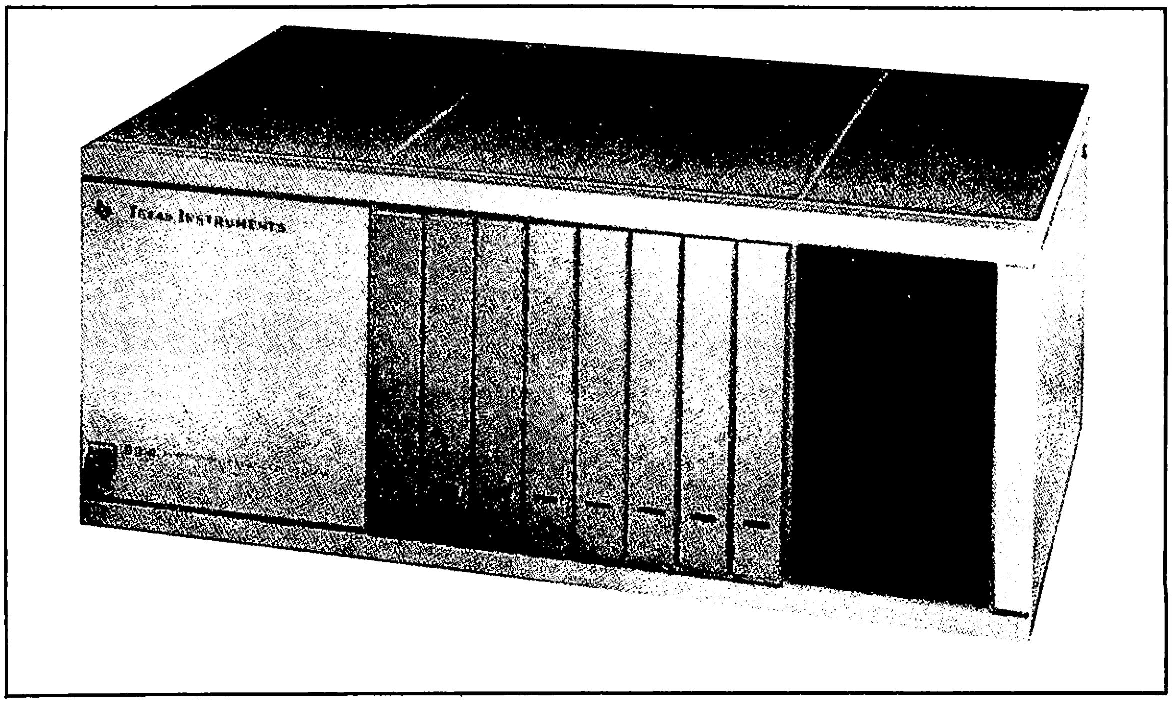


Fig. 1-12. The peripheral expansion system from Texas Instruments.

pansion system attached to the TI-99/4A, you can quickly change computer capabilities by adding different accessory cards. You can also install a TI disk drive in the portion of the compartment designed for this purpose. To access the interior of this accessory, remove the top of the unit and slide in the accessory cards. The system can hold up to seven accessories, including the disk drive controller card, the RS-232 interface, the TI memory expansion card, and several other accessory boards. To handle the increased power drain of the many options this device can hold, a separate 150 watt power supply is provided. The unit weighs about 20 pounds and is operated from the ac line.

Disk Memory System

The TI disk memory system is a combination of hardware and software that allows you to store and retrieve data on single-sided or double-sided disk measuring 5¼ inches in diameter. Disk systems accomplish the same thing as cassette tape storage systems, but faster. Each single-sided disk holds over 700,000 bits of information; a double-sided disk holds nearly 1,500,000 bits. Thus, the single-sided disk has a holding capacity of about 90K bytes, and the double-sided disk with this system will hold twice this amount.

The disk memory system consists of a disk controller card, disk memory drive, and the disk manager command software. The disk controller card tells a disk drive where to position the magnetic head in order to read or write information properly. The controller also puts an index on the disk, making the data that has been written easy to locate. It can control up to three disk memory drives.

The disk drive spins the disk at a constant speed and controls the movement of the magnetic head. There is a special compartment in the peripheral expansion system for installation of one TI disk memory drive.

The disk manager solid state software command module helps you maintain the information on your disks. Naming and renaming disks, renaming files, deleting files, copying files, and copying disks is done with the disk manager module.

Because the control software needed for the disk system is in permanent ROM, in the disk manager command module, and in the controller, the disk system uses a relatively small amount of working space in the computer's available memory (RAM).

Memory Expansion Card

The TI memory expansion card adds 32K bytes of random-access memory to the 16K bytes of RAM resident in the TI-99/4A console. The expanded memory is designed for use with TI Extended BASIC and other languages contained on the command module. The memory expansion card attaches to the peripheral expansion system and requires that TI Extended BASIC or another specialized command module be inserted in the computer console. Most software packages cannot make use of the memory expansion card without the addition of Extended BASIC or some other special command module.

Telephone Coupler (Modem)

The Texas Instruments telephone coupler (Fig. 1-13) enables your TI-99/4A to send and receive messages through a standard tele-

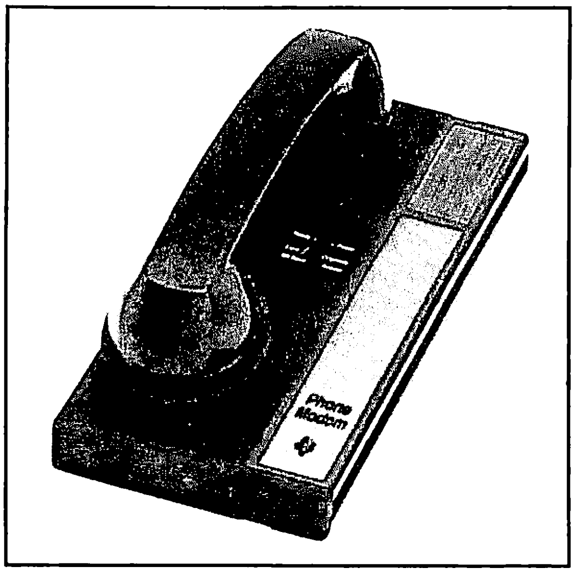


Fig. 1-13. The TI telephone coupler enables the TI-99/4A to send and receive messages via a standard telephone.

phone. Use of the telephone coupler requires an RS-232 interface unit.

The telephone coupler functions as a modulator to convert the data you enter on the console into signals that can be sent over telephone lines. It also functions as a demodulator to convert data received over telephone lines back to its original form. Using the telephone coupler is simple. It is powered by a UL-listed low-voltage transformer, which is included. A cable connects the coupler to the RS-232 interface. A standard telephone headset inserts into the flexible acoustic couplers on the telephone coupler. This device may be used with many RS-232 compatible terminals or computer systems for communication over standard telephone lines.

The telephone coupler offers two basic modes of operation, called the originate mode and the answer mode. In the originate mode, you are the party who begins all communications with the remote terminal. In the answer mode, the remote terminal originates communications. This device is capable of transmitting at a data rate that is continuously variable up to 300 bps.

Cassette Interface Cable

This interface (Fig. 1-14) cable plugs into the TI-99/4A console and allows you to connect one or two cassette recorders to the computer. I stated earlier that the basic TI-99/4A package is complete, in that it allows you immediately to begin writing and running computer programs, providing you have a television or monitor. With the standard package, however, you cannot store any programs, even if you have a storage device such as a cassette tape recorder. I assume that TI includes the video modulator with their basic package on the assumption that most homes have a television and that a receiver is necessary to use the computer. It's my feeling that most homes also have cassette tape recorders; and therefore,

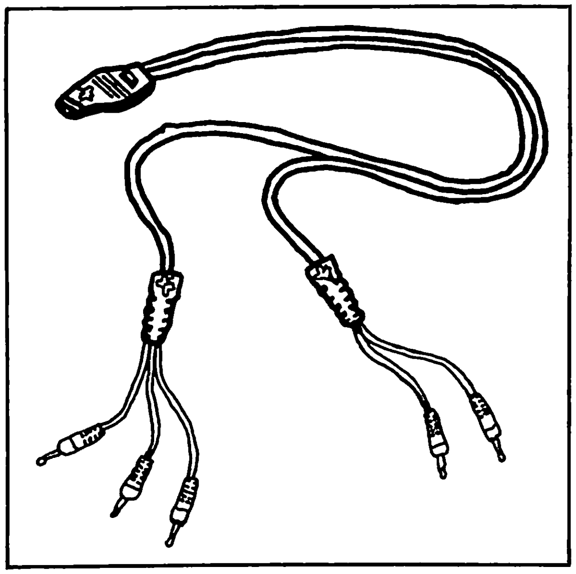


Fig. 1-14. The cassette interface cable turns a cassette recorder into a memory storage device for the TI-99/4A.

the cassette interface cable should be provided as part of the base package to allow for the saving of programs. I guess I'm especially touchy about this particular cable, since I incorrectly assumed that one was included with my purchase of the basic console. When I returned home after a long drive, I found that the cable was an option and it was impossible to locate a substitute locally. Although disk users would not need a cassette interface cable, I think the majority of TI-99/4A owners will use cassette storage since a disk drive costs more than the computer itself.

With the cassette interface cable, you can use one or two recorders to save and load computer programs. Texas Instruments points out that the use of two cassette recorders is especially helpful for programming applications where a lot of memory space is required. Many cassette recorders can be used with the computer, although each should be equipped with a separate volume control, tone control, microphone jack, remote jack, earphone or external speaker jack, and a digital tape counter. The latter is not mandatory, but is a tremendous help in locating the correct tape position for a particular program.

To connect the computer to the cassette recorder(s), insert the 9-pin D connector into the 9-pin outlet on the rear of the computer console. This is the outlet directly to the left of the power cable outlet when facing the back of the unit. On the other end of the cable, the plug with the red wire goes to the microphone jack. The one with the black wire goes to the remote jack and the third one connects to the earphone jack. In most cases, the second set of cassette recorder plugs are not used, so these simply hang free.

Texas Instruments includes a list of recorders from various manufacturers whose products are known to work with the TI-

99/4A. This does not include all of the recorders that work, and indeed, most types can be made to work. Some of the inexpensive recorders do not have a tone control, so it may be necessary to adjust the volume to make up for this.

There is one point that should be known. Most cassette recorders operate from internal batteries as well as from house current. I would shy away from the use of battery power when saving and loading computer programs. As the batteries deteriorate, motor speed will slow, and information may be erratically recorded or output to the computer. In many instances, replacing the batteries with a fresh set will correct this; in one instance, it may not. A set of weak batteries in the recorder causes the motor speed to slow up and the tape is pulled across the record head at a slower than normal rate. You may successfully save (record) the program on tape. With a new set of batteries, the motor speed will pick up to normal again, but the program that was saved while the other set of batteries was in place was recorded at a slower speed. With the fresh set, the playback of this recorded program is faster than intended. This can be disastrous, and you may not be able to retrieve the program.

Also when batteries become weak, motor speed may fluctuate. The tape may travel across the record head for a few minutes at one speed and for a few more at a faster or slower speed. This is an even bigger problem and almost assures that you can never retrieve the program. The same thing can happen when ac power is used to drive the recorder, but only when there is a defect in the recorder circuitry. I highly recommend the use of an ac power supply for recording programs.

If you decide to use batteries, make absolutely certain that fresh batteries are installed at appropriate time intervals. You may wish to use rechargeable batteries that can be recharged from the ac line after every usage.

Cassette storage is slow compared with disk storage, but it's also quite inexpensive and the data is stored quite accurately. From a price standpoint, it is the most efficient data storage medium available today. For most owners of the TI-99/4A, cassette storage will be completely adequate.

Cassette Program Recorder

Texas Instruments announced early in 1983 a new, compact cassette program recorder, (Fig. 1-15) designed for use with the TI-99/4A. The recorder package includes a computer interface cable for the TI-99/4A.

Features of the unit include the ability to be controlled from the TI-99/4A, an automatic recording level control (ALC), a digital tape counter, clearly marked optimum settings for volume and tone control, color-coded input jacks for easy setup, a pause control, and a built-in condenser microphone. The program recorder, with a suggested retail price of $69.95, can operate either on four C batteries or on ordinary ac power through the included cord.

Speech Synthesizer

The Texas Instruments solid state speech synthesizer (Fig. 1-16) makes possible the exciting addition of speech to the TI-99/4A. The speech synthesizer requires an optional command module preprogrammed for speech, such

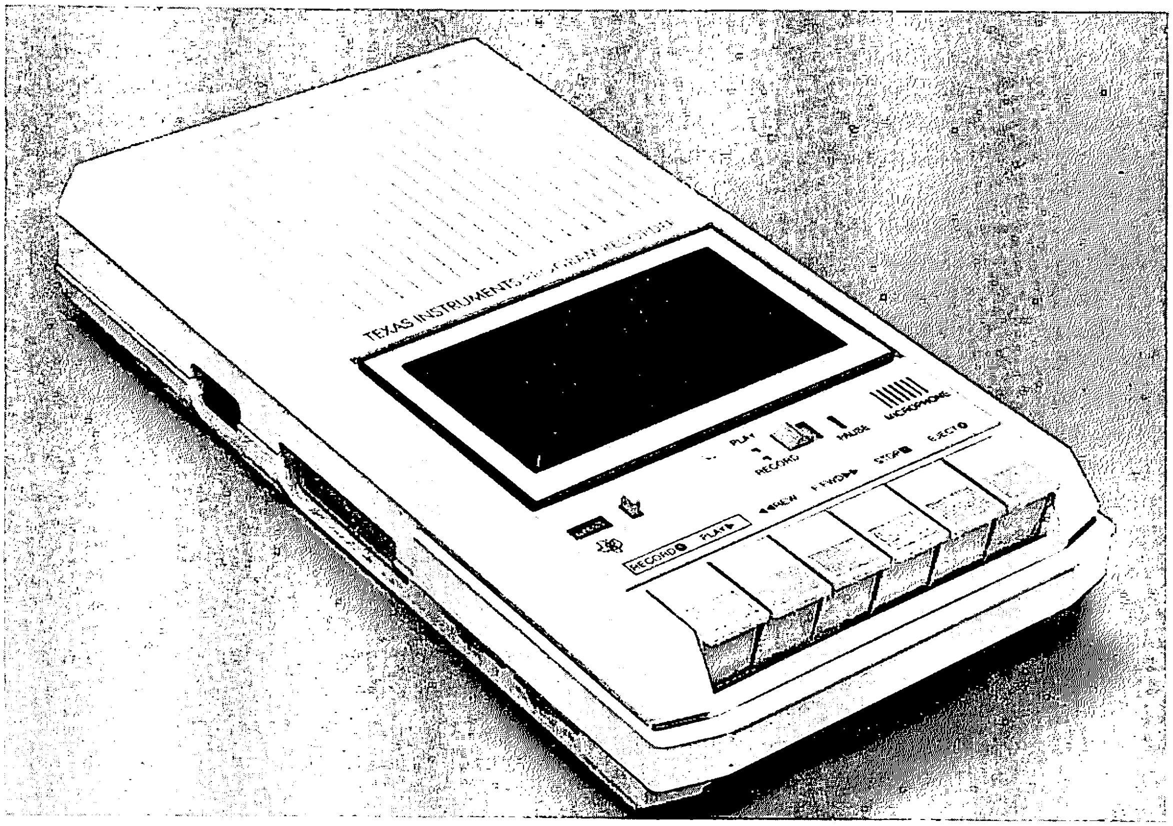


Fig. 1-15. The cassette program recorder from Texas Instruments sells for about $70.00.

as the speech editor command module. These preprogrammed modules allow the speech synthesizer to be used without the need to do any programming. Speech can also be included as part of your own programs in TI BASIC.

The speech synthesizer is entirely electronic. There are no taped voice recordings or any other traditional recording medium. A vocabulary of words and phrases is permanently stored on chips contained within the speech synthesizer. Each word has been transformed into a pattern of bits. When processed, each pattern drives electronic circuitry that rebuilds the requested word and audibly reproduces it through a loudspeaker. The speech synthesizer contains a resident vocabulary of over 300 words. Capacity is expandable with optional plug-in speech modules.

The synthesizer docks into the TI-99/4A by means of built-in connectors. Insert one of the command modules designed to call up speech from the device, and you are ready to go.

The speech synthesizer provides a voice for the computer, creating many new applications and enhancing the effectiveness of existing ones. It can communicate with you even if you are not near the display. It can recite instructions to those unable to read, or where written instructions might interfere with the display. It can provide exciting comments and sound effects in games, and it can reinforce concepts in educational applications.

The speech synthesizer can be used in several ways. In one mode, it is controlled by a command module other than the speech editor command module. This must be a command module with speech capability.

All other methods of operation require the use of the speech editor command module itself. Using TI BASIC, words, phrases, or sentences may be recited under program control.

The speech editor command module can also immediately recite words, phrases, and sentences without your having to write a program. In this mode, just type in the desired word, push the enter key, and the speech synthesizer says the word. Figure 1-17 provides a listing of the device's resident vocabulary.

Optional plug-in speech modules add additional vocabulary, in specialized areas. Drop the appropriate speech module into the speech synthesizer and the new words are immediately accessible to you, in addition to the resident vocabulary in the speech synthesizer itself.

Impact Printer

The TI impact printer (Fig. 1-18) is a fairly new offering for the TI-99/4A. The printer itself has been out for a long time, because it's manufactured by Epson (probably the best-known manufacturer of computer printers in the world). The Epson printer, the MX-80, is almost a standard in the personal computer industry. The IBM Personal Computer, for instance, uses this same printer (with IBM's name on it).

This printer is capable of producing 80 characters per second and can handle 40-, 66-, 80-, and 132-column widths. It can print text or

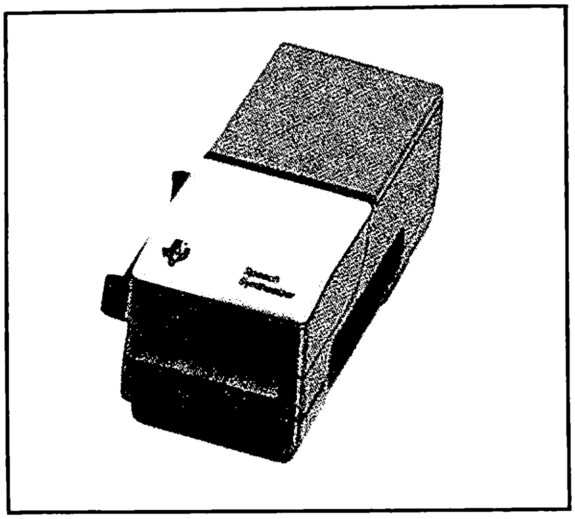


Fig. 1-16. The solid state speech synthesizer made by TI adds a voice to your TI-99/4A.

+ (positive)	but	draw	gives	it
– (negative)	buy	drawing	go	j
• (point)	by	e	goes	joystick
0	bye	each	going	just
1	c	eight	good	k
2	can	eighty	good work	key
3	cassette	eleven	goodbye	keyboard
4	center	else	got	know
5	check	end	gray	l
6	choice	ends	green	large
7	clear	enter	guess	larger
8	color	error	h	largest
9	come	exactly	had	last
a(ā)	comes	eye	hand	learn
a1 (ah)	comma	f	handheld unit	left
about	command	fifteen	has	less
after	complete	fifty	have	let
again	completed	figure	head	like
all	computer	find	hear	likes
am	connected	fine	hello	line
an	console	finish	help	load
and	correct	finished	here	long
answer	course	first	higher	look
any	cyan	fit	hit	looks
are	d	five	home	lower
as	data	for	how	m
assume	decide	forty	hundred	made
at	device	four	hurry	magenta
b	did	fourteen	i	make
back	different	fourth	I win	me
base	diskette	from	if	mean
be	do	front	in	memory
between	does	g	inch	message
black	doing	games	inches	messages
blue	done	get	instruction	middle
both	double	getting	instructions	might
bottom	down	give	is	module

Fig. 1-17. The resident vocabulary in the TI speech synthesizer.

more	point	shape	that is incorrect	up	you
most	position	shapes	that is right	upper	you win
move	positive	shift	the1 (thĕ)	use	your
must	press	short	the (thē)	v	z
n	print	shorter	their	vary	zero
name	printer	should	then	very	
near	problem	side	there	w	
need	problems	sides	these	wait	
negative	program	six	they	want	
next	put	sixty	thing	wants	
nice try	putting	small	things	way	
nine	q	smaller	think	we	
ninety	r	smallest	third	weigh	
no	randomly	so	thirteen	weight	
not	read (rēd)	some	thirty	well	
now	read1 (rĕd)	sorry	this	were	
number	ready to start	space	three	what	
o	recorder	spaces	threw	what was that	
of	red	spell	through	when	
off	refer	square	time	where	
oh	remember	start	to	which	
on	return	step	together	white	
one	rewind	stop	tone	who	
only	right	sum	too	why	
or	round	supposed	top	will	
order	s	supposed to	try	with	
other	said	sure	try again	won	
out	save		turn	word	
over	say	t	twelve	words	
p	says	take	twenty	work	
part	screen	teen	two	working	
partner	second	tell	type	write	
parts	see	ten	u	x	
period	sees	texas	uhoh	y	
play	set	instruments	under	yellow	
plays	seven	than	understand	yes	
please	seventy	that	until	yet	

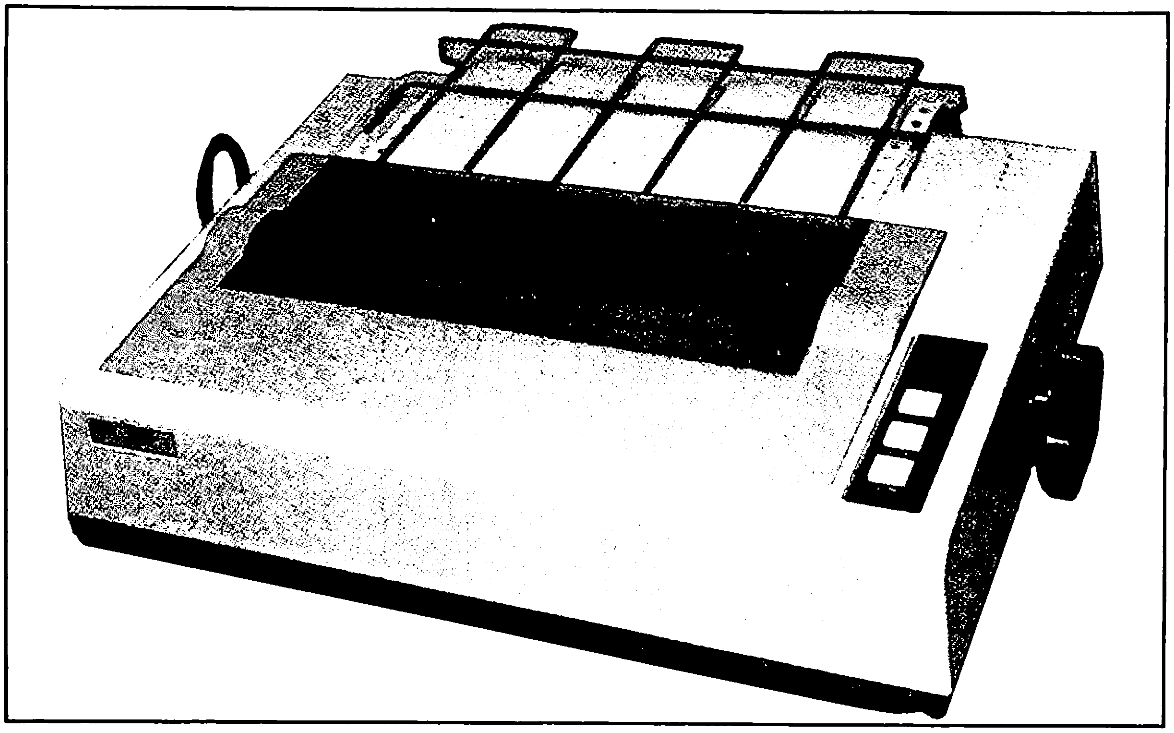


Fig. 1-18. The TI impact printer offers a speed of 80 characters per second and a maximum width of 132 columns.

graphic data. This is a bidirectional printer, which means it prints from left to right and then from right to left. There is no nonprinting return stroke. The first line of a page of text is printed from left to right, just like on a typewriter. However, the second line will be printed from right to left in reverse order.

This printer does not require special paper. It features a 9 by 9 dot matrix print head that can be easily replaced. A single-unit ribbon cartridge is easily inserted, and you can choose from several different ribbon colors.

Connecting the impact printer to the TI-99/4A requires the RS-232 interface and the printer cable supplied by Texas Instruments.

It produces excellent quality hard copy printouts, but is not a letter-quality printer. Letter-quality printers are used in word processing operations that require all letters and documents to appear as if they were typed on a high-quality typewriter. Because most letter-quality printers have typewriter-like mechanisms to do the actual printing, they are usually slower than dot matrix printers.

The TI impact printer produces neat and perfectly readable copy. The type will not appear to be as perfect as that produced by a good letter-quality printer or for that matter, a good typewriter, but because the TI-99/4A is not designed for sophisticated word processing (in my opinion), a letter-quality printer should not be required.

Cartridge Storage Cabinet

If you collect a lot of TI-99/4A software, you've got to store the cartridges or cassettes when they're not in use. Figure 1-19 shows the TI storage cabinet, which sells for about $15.00. The cabinet holds 12 cartridges or cassettes in two sliding doors. The case is designed to be stackable, so two or more may be combined vertically to increase storage capability.

Compatible Computers

The next two devices may not be considered options to the TI-99/4A, but they are available and they can be interfaced with this computer. Since these are relatively new announcements from TI, it is appropriate that they be mentioned here.

TI-99/2 (Fig. 1-20) is a new computer from Texas Instruments that is believed to be the first 16-bit computer for less than $100 in the world. Unlike most computers in this price range, the TI-99/2 Basic Computer uses software on solid state cartridges as well as on cassettes. In addition, Texas Instruments is introducing low-cost peripherals and software for the TI-99/2 that will also work the TI-99/4A.

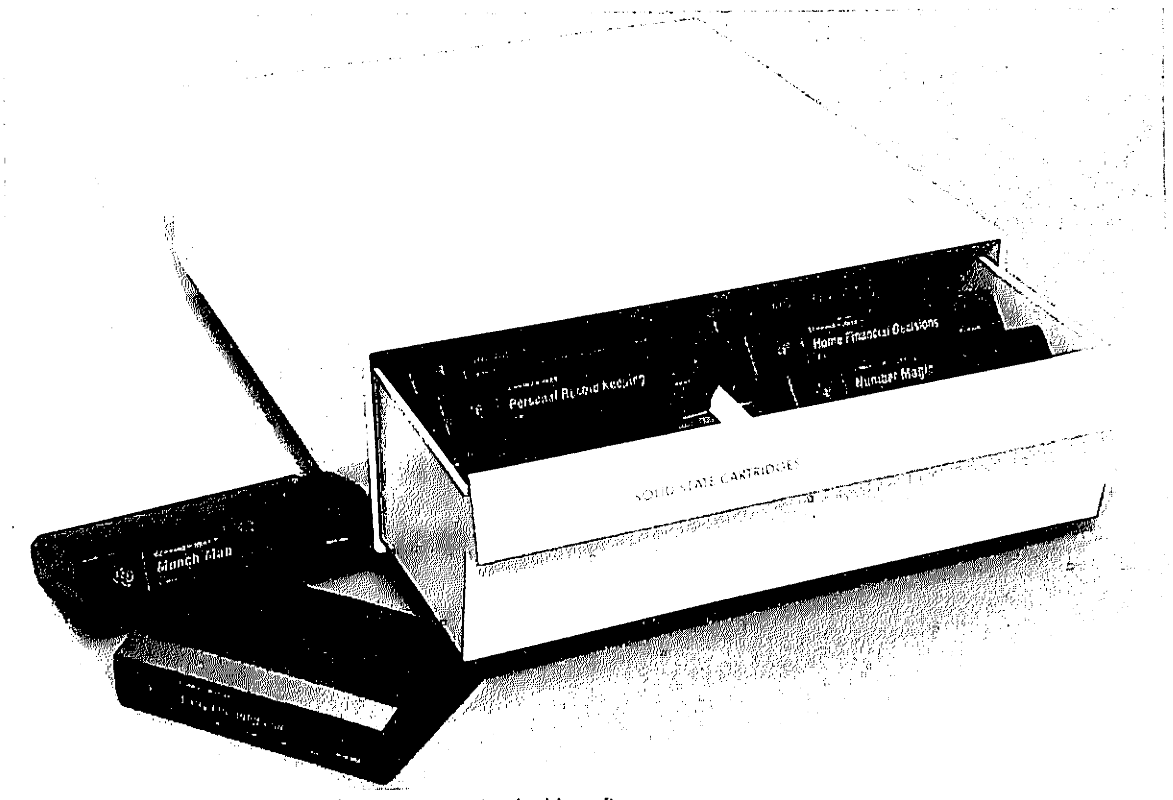


Fig. 1-19. A cartridge storage cabinet can protect valuable software.

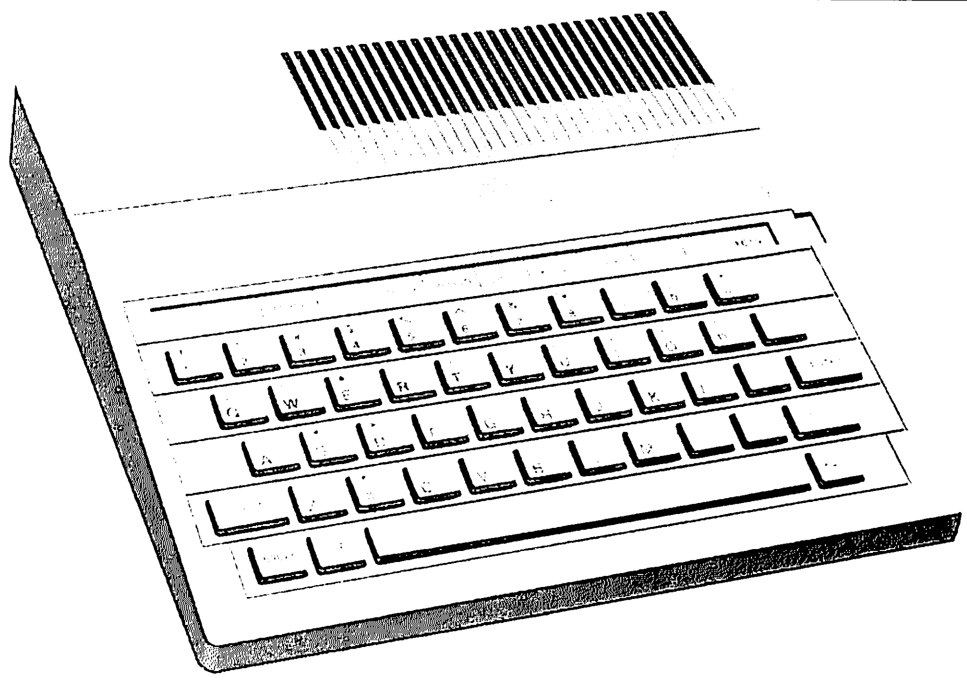


Fig. 1-20. The TI-99/2 computer. (Courtesy Texas Instruments Inc.)

William Turner, President of the Consumer Group said that the TI-99/2 is designed to allow computer novices to learn to program a computer in TI BASIC and BASIC-supported assembly language. This machine is targeted primarily at the technical enthusiast, engineer, or student. He stated that TI expects this computer to be purchased as the first computer in the home for persons just beginning their computer experience, or as a second computer.

The TI-99/2 console has an elastomeric typewriter-like keyboard with raised keys in a staggered QWERTY arrangement similar to the TI-99/4A. The computer has 4.2K bytes of built-in random access memory (RAM) and can be expanded to a total of 36.2K bytes of RAM.

Most peripherals for the system plug into the Hex-bus peripheral interface connector in the rear of the console. The Hex-bus port allows users to connect any peripheral developed for the TI compact computer family. Currently, these consist of the RS-232 interface, HX-3000; the Wafertape digital tape drive unit, HX-2000; and the HX-1000 four-color printer/plotter. Other peripherals such as modems, printers, a wand input device, and a black-and-white television monitor are scheduled to be available in late 1983.

Two solid state software cartridges, Learn to Program and Learn to Program BASIC, are available for the unit. Other cartridges will be available as well. Suggested retail price for the cartridges is $19.95.

Twenty software programs will be available as well. Educational programs include: Picomath-80, Math I and II, Statistics I and II, Sunrise Time, Datetimer, and Civil Engineering. Programs for personal management are: Household Formulas, Checkbook Manager, Purchase Decisions, and General Finance. Entertainment cassettes include: Lunar Landing, Bioplot, The Minotaur, TI Trek, and Mind Games I, II, III, and IV. Picomath-80 is priced at $19.95; all others are $9.95 each. These programs, and all user-written programs, can be run on the TI-99/4A home computer.

The TI-99/2 features monochrome display capability and contains a built-in RF modulator. The included video cable and antenna switch are used to connect the computer to any television set. A cassette interface cable is also included to interface directly to the new TI Program Recorder or another cassette tape player. In addition, the TI-99/2 comes with an ac adapter, a user's manual, and a demonstration cassette.

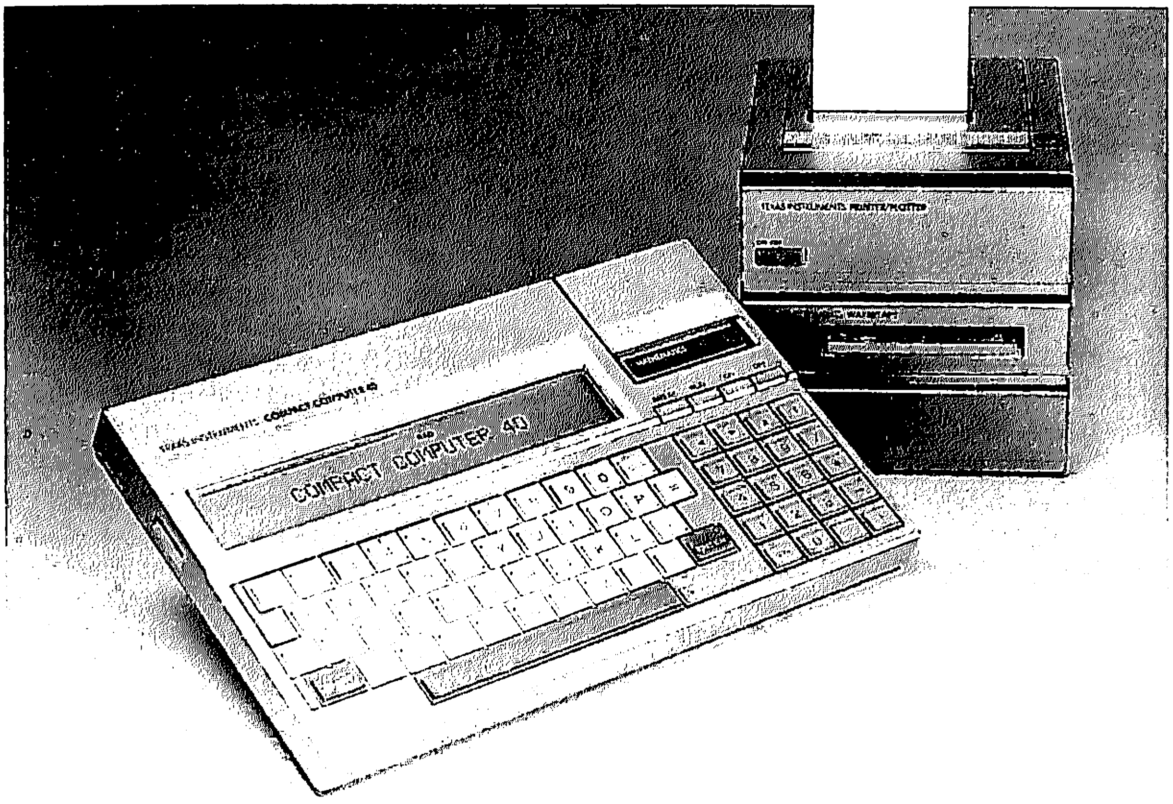


Fig. 1-21. Compact Computer 40 (CC-40) is the first member of a new series of computers from TI which are small but designed for professionals. (Courtesy Texas Instruments Inc.)

The Compact Computer 40 (CC-40) was announced on January 6, 1983. It is the first member of a new series of small computers designed for professionals. Shown in Fig. 1-21, the computer is similar in appearance to the TI-99/2, but includes a numeric keypad and a built-in liquid crystal display. The CC-40 is programmable in Enhanced BASIC and can run preprogrammed applications software loaded from plug-in solid state cartridges or from small tape cartridges.

The system is battery-operated and fits unobtrusively on a desk or into a briefcase. It is designed to be used as a small personal desktop cordless computer and for communications. Its small size and battery operation also provide extensive capability for portable computer applications.

The computer console has a 34K byte ROM that contains a BASIC language interpreter allowing operation in BASIC. The BASIC language built into the CC-40 is compatible with TI BASIC. Calculator functions are available. The computer contains 6K bytes of RAM and can be expanded to 16K bytes. The CC-40 has a suggested retail price of $249.95.

A plug-in module port is provided for application software of up to 128K bytes of ROM. This port can also be used to expand the random-access memory of the computer. The back of the console houses a Hex-bus intelligent peripheral interface connector, allowing connection of any Hex-bus compatible peripherals, as well as future TI products.

Three low-cost peripherals will also be available: an RS-232 interface, a printer/plotter, and a Wafertape digital tape drive. Other peripherals such as a wand input device, modems, printers, and a black and white television monitor should be available late in 1983. Each peripheral includes a Hex-bus port and interface cable. Peripherals will also operate with the TI-99/2 and, with an adapter, will work with the TI-99/4A computer as well.

The RS-232 interface allows direct connection to serial-input printers and modems. With the addition of an optional cable, the interface can connect to a parallel-input printer. The RS-232 interface, HX-3000, has a suggested retail price of $99.95.

The printer plotter is an x-y plotter with four-color capability using 2½ inch wide plain paper. In addition to x-y plotting, it can print up to 36 characters per line. The printer/plotter peripheral, HX-1000, has a suggested retail price of $199.95.

The Wafertape digital tape drive can store up to 48K bytes and has a data transfer rate of 8,000 bits per second. The Wafertape unit, HX-2000, has a retail price of $139.95.

Twenty-two software applications packages, including 8 plug-in solid state software cartridges and 14 Wafertape cartridges, are also available. The plug-in cartridges, which sell for prices ranging from $39.95 to $124.95, are: Mathematics, Finance, Perspective Drawing, Statistics, Business Graphics, Nonparametric Statistics, and Advanced Electrical Engineering ($59.95 each); Editor/Assembler ($124.95); and Games I and Games II ($39.95 each). Wafertape cartridges, which have a suggested retail price of $19.95 each, are: Elementary Dynamics, Regression/Curve Fitting, Pipe Design, Production and Planning, Inventory Control, Electrical Engineering, Thermodynamics, Photography, Solar Energy, Profitability Analysis, Quality Assurance: Sampling Plans, and Quality Assurance: Control Data. A total of 75 applications solutions cartridges (48 solid state and 27 Wafertape programs) are also available. TI is initiating aggressive third-party authorship programs as well as developing software internally.

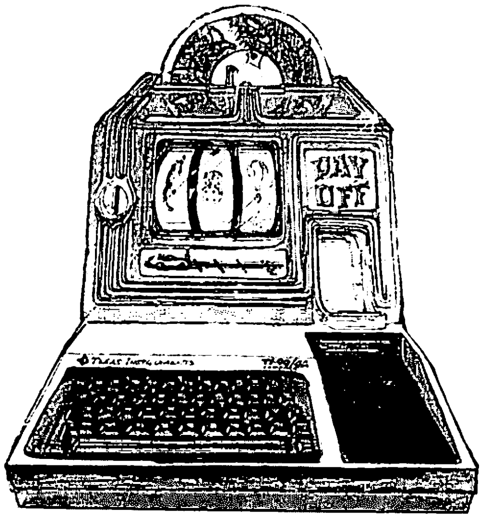
The CC-40 console is 9½ inches by 5¾ inches by 1 inch and weighs 22 ounces. The display is a scrollable 31-character liquid crystal display (LCD) capable of displaying upper- and lowercase characters. In addition, there are 18 built-in indicators for user feedback including shift, control, function, degrees, radians, grads, and 6 user-settable flags.

The keyboard has a staggered QWERTY key arrangement with a numeric keypad. Key spacing allows for easy key entry without making the unit excessively large. A tilt stand is built into the back of the console to provide an optimum viewing and keying angle.

Four AA alkaline batteries provide power to the console for up to 200 hours. Memory contents are retained even when the unit is turned off. The unit may also be connected to a standard 115-volt ac power outlet using an optional adapter, AC9201, available for $14.95.

Texas Instruments continues to offer new products and accessories, but usually attempts to provide methods of interfacing them with previous offerings. This speaks well of the company and assures that any product you purchase does not suddenly become antiquated by the introduction of a new product.

Chapter 2



Game Programming Concepts

Programming games on the TI-99/4A computer opens up a whole new world of computer enjoyment. The more you program, the more you begin to visualize almost everything in programming terms. This statement applies to mathematical problems, as well as everyday experiences. It has been stated that everything in the universe can be explained by mathematics. While there is no way of proving this statement, it is certainly very safe to say that almost everything can be explained mathematically.

THE IMPORTANCE OF MATHEMATICS

Mathematics is a language which is based upon collective human knowledge. It is a method of describing physical objects, functions, and processes. Since these are highly complex in nature, one branch of mathematics may be used to describe a certain attribute, while another branch is used to describe another. Mathematics is not a fixed language, but one that changes as human knowledge changes. When new facts are discovered and new mathematics developed, this new branch does not necessarily make the old branch obsolete. Rather, the new branch becomes an extension of our mathematical language.

A microcomputer is a mathematical instrument. All programming describes mathematical formulas, often in common language terms. Successful computer programming involves a sophisticated use of mathmatics. Therefore, one cannot fully evaluate a situation and turn it into a computer program without an understanding of mathematics.

The computer programmer must learn to think in mathematical terms. If a game program is to include the figure of a man running across the screen, all of the motions must be described mathematically and input to the machine accordingly. This conversion may sound very complex, but in most cases, it is not as you will see in the following chapter on TI graphics.

Many of the games (in fact, all of them) that have been enjoyed by persons for many hundreds or even thousands of years can be mathematically described. Therefore, they may be committed to the computer. Knowing this, let's take a game and input a mathematical description of it to the computer. Tic-tac-toe is an excellent example of a fairly simple game. Because this is a simple game as far as our human computer (the brain) is concerned, you might think that committing it to a computer program would be an equally simple task. Well, it's not.

First, it is necessary to draw the crosshatch on the screen. This is accomplished by establishing the mathematical coordinates of each line. These tell the machine where the beginning and end of each line is to be. Once the game board has been drawn, it is necessary to establish mathematical coordinates for the Xs or Os that are to be placed within the squares and also to establish a method of specifying which box a given X or O should be placed in. In most computer tic-tac-toe games, the X is assigned to the human player, while the O is assigned to the computer. The human chooses the square he wishes to fill by inputting the number of the square via the keyboard. The number in itself means nothing to the computer, but a mathematical interpretation of this human input is provided by statements within the program. This allows the machine to internally interpret the desires of the human, converting them to a mathematical formula which, in turn, brings about a specific screen function, the placement of the character in the correct box.

So far, there's been little problem. However, the computer must now simulate human thinking by deciding what move to make based upon the human's moves. In fact, the machine does not simulate human thinking; it will simply carry out programmed instructions, which are the product of human thinking. Simulation, though, is not a totally incorrect description here, because many of the human thinking processes involved in playing a game of tic-tac-toe have been converted to mathematical form and input to the machine in the BASIC language. The computer can detect when a win is about to occur on the human's part and will take steps to block the win if possible.

As an example of this, let's assume that the human is about to win diagonally. Assuming that the squares are numbered horizontally from one to nine (Fig. 2-1), this would mean that the human has filled squares one and five. Assuming that the S$ array reflects the Xs and Os that have been assigned to the squares, the

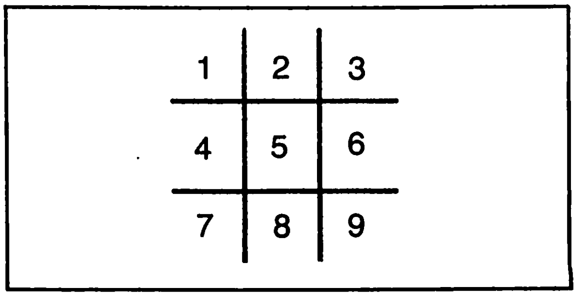


Fig. 2-1. The numbered tic-tac-toe grid.

computer program portion that protects against a win in this situation might read:

```
10  IF S$(1) = "X" THEN 20 ELSE
    (Branch to another portion of pro-
    gram)
20  IF S$(5) = "X" THEN 30
30  S$(9) = "O"
40  PRINT S$(9) (in square 9)
```

This allows the computer to block at square 9, provided the conditions are met. If two human beings are playing the same game and the one who has the next move is confronted with the same possible win situation, his mind acts in much the same manner. To put this into a crude program form, the human equivalent might be

SQUARE 1 = X AND SQUARE 5 = X
SO SQUARE 9 HAD BETTER EQUAL 0

The human brain has been programmed since birth and acts upon all information that has been input. The human purpose here is to win the game based upon a collection of data. The computer, however, has no purpose. It is simply there to carry out human instructions . . . but the human that programmed the tic-tac-toe game was converting his natural programming into computer terms. The programmer wants the computer to take his place and act upon his store of knowledge about the game in order to effect a win.

Humans often pull from a vast store of knowledge. With present programming languages, it is necessary to go all the way back to basics in order to decipher just why we do the things we do. Most of these actions are based upon logic, even though some of the data will undoubtedly be faulty. We can only get the computer to do what we would do using our own logic, given a certain circumstance. In programming tic-tac-toe and all other games, we must also figure out all of the possibilities for every move. These possibilities increase to maximum complexity after a set number of moves. The computer must determine which moves have been made by the human and by itself and then effect the most logical step to prevent a win by the human or to bring about a win by itself.

When playing tic-tac-toe, the human thinks ahead, while the computer can only react to a human's moves based upon programming. Certainly, it's quite possible to allow the machine to make a good guess as to what the human's next move will be based upon its own move. To create an accurate anticipatory program section, it would be necessary for the programmer to play many different humans (a broad cross-section) a large number of times, record and analyze their moves, and utilize this information in the game program.

SELECTING GAMES TO PROGRAM

It is not necessary to dream up computer games out of your head, although this can be a very rewarding experience. As a start, however, it may be wise to use traditional games that have been played without the use of a computer. Your knowledge of these games will aid you in writing programs to simulate them on the machine. Take Spin The Bottle, for example. This would be a fairly simple program to write. You could use a line with an arrow instead of a bottle, which would be difficult to draw on the screen. Coordinates can be worked out for various line positions and committed to a loop that was randomly controlled. Upon initiating the program, you might be asked to input the names of two or more players. These names would appear at two or more positions on the screen. When the loop is engaged, the arrow would spin and stop randomly at any of the possible positions. What the humans choose to do with the screen results is entirely up to them (or to you, as the programmer).

For the most part, computer games are fairly simple. The complexities are brought into the program by the necessity of simulating human responses. In other words, the computer must frequently respond in a human manner. This is often handled with messages such as OOPS, YOU MISSED THAT ONE, HA HA, I BEAT YOU AGAIN, OR OH NO, YOU BEAT ME. These messages are totally unnecessary for the actual mechanics of playing the game, but they are the human elements that can make the difference between a successful game and one that is rather drab and boring.

In many instances, these messages require far more program lines than the basic game itself. Many computer games can be basically programmed in twenty lines or less. However, a hundred or more lines may be used in writing the program in order to add messages and even simple graphic displays to make the game more interesting. Take a dice program, for example. Many computer games that simulate the roll of dice simply print two separate numbers on the screen, each chosen at random and ranging from a value of 1 to 6. Each number represents a die. This is totally unnecessary in most dice games, since the sum of the two die values is what the game revolves around anyway. When a dice program is simulated on the computer, it could just as easily involve a single random number which could range in value from 2 to 12. This type of display would be far more practical . . . but it probably wouldn't be as interesting.

PLANNING YOUR COMPUTER GAME

From a realistic standpoint, we humans do not really play computer games. Rather, we play and enjoy simulations of games on the computer. Most of the popular computer games today simulate board games and even childhood action games that have been with us for hundreds or even thousands of years. Let's face it. Even the most sophisticated graphic space wars game is basically a computer simulation of "Cowboys and Indians," which many of us played as kids. Many of today's computer games are similar to the mechanical arcade games of a few decades ago where moving ducks were knocked over by pellet guns or actual firearms. Almost any game which involves a target and a projectile can be eventually traced to ancient games or pursuits.

Adventure games are based upon true human experiences or modifications of them. For example, many of us would love to have the opportunity to go on an African safari and attempt to bag dangerous game. We like the idea or the adventure aspects of such a fantasy. However, when you get right down to it, few of us can really afford such a trip. Also, Africa is hot, dusty, and some of those animals really are dangerous. However, a computer game, which involves the same elements in fantasy form, can occupy the mind for quite some time and simulate the adventure that the actual event might provide without any of the discomforts or dangers of actually being there.

From a computer game standpoint, the attention which must be given to a game relates directly to its popularity of same. Why do people play games? To relax? Relaxation is certainly an element, but is this really true of action computer games? Not really. When you get involved in a game that requires 100 percent of your attention, you cannot relax. You must be tense and ready for anything to happen. This indirectly relaxes many persons. This is due to the fact that good action games allow you to escape. When you're bored with the humdrum world, something that requires 100 percent of your attention takes your mind off your problems. There is a goal in mind, a goal which you want to attain as a player. However, unlike true life situations, if you don't attain that goal, it's no big deal. You can just come back a bit later and try again. In a few countries where the populous must travel almost exclusively by foot, fatigue from walking is often handled in what we would consider an unusual manner. After walking for several miles, a person might rest and relax by running for a few minutes. While this sounds unusual to us, it does work and is borne out by scientific fact. When one's muscles become fatigued from walking, running is an excellent remedy because many different muscles are used. By alternating between the two modes, one gets to the final destination in the shortest period of time without undue physical strain.

Computer game programs relax players in a similar manner. They allow the player to function on a different mental level, which may be more demanding, but only for a short period of time. Additionally, the player has the option of quitting at any time. This option is not often afforded in true life situations. In all cases, the player is in full control of the efforts that are put forth.

However, a computer game must be attractive enough to warrant the player's full attention. This is where programming difficulties usually arise. The player's imagination must be stimulated. To do this, programmers pull from experiences that most humans have been subject to, either directly or through the media. This is why space games are so popular today. Several decades ago, the same type of games that today involve space invaders and flying saucers may have involved charging guerillas and bush fighters. Both types of games are played in the same manner; i.e., blow away your target, but each is aimed at the imagination of the players of their respective decades.

A successful computer game must present a sizeable difficulty factor, but must not be so difficult as to be practically impossible. How difficult to make the game will be decided by many different factors, chief among which is the age or skill range of the players. Undoubtedly, you've noticed that many of the home video games offer a wide range of difficulty levels. This is because such games may be played by small children as well as adults. Varying difficulty levels can often be built into computer game programs with a few extra program lines. Therefore, whenever writing a computer game program, it is mandatory that you know the age and general skill levels of those who will be playing it.

The ideal computer game will offer varying degrees of challenge in order to be used by a large number of players who will undoubtedly exhibit different levels of proficiency. If a game is too easy, it quickly becomes boring and is set aside. By the same token, if a game is too difficult, frustration quickly sets in and it is thrown on the same scrap heap. For this reason, the game programmer will do well to learn from the commercial games that are already popular and let them partially determine what is to be written in program form.

SPACE AGE GAMES

In recent years, space games, which come in many different forms, have become quite popular. The graphic renditions may display a deep space background lit with stars. The foreground often contains two different types of space vehicles that may have to navigate among asteroids or try to blow each other out of existence. These programs often involve quite a bit of graphics. The program usually starts out by creating a dark blue background, which is then dotted with tiny points of light. A wide range of colors may be chosen for the points that represent stars. The drawing of the space vehicles on the screen is not terribly difficult either. For example, the Starship Enterprise of Star Trek fame may be graphically represented quite easily using the CALL CHAR subprogram. Klingon vessels may be represented in much the same manner. The attack with phasers and photon torpedoes can be enhanced by sound effects using the CALL SOUND subprogram.

Now, let's look at such a program mathematically. A call screen statement is used to set up the initial blue background. We could determine the coordinates of each point on the screen that will be used to represent stars. However, the easiest method is to set up minimum and maximum coordinates and use the RND function to place a large number of dots on the screen with a minimum number of program lines. When a photon torpedo is fired, its originating point is based upon the coordinates of the attacking ship at that instant. The torpedo may continue to the edge of the screen or may be limited in range by a maximum coordinate figure. In any event, when the screen coordinates of a fired photon torpedo are the same as the coordinates of the target, a hit is registered. Depending on the graphics, the target may disappear or explode into many different pieces. Using the latter, the angle each piece travels from its exploding point is determined by more coordinates. Causing the struck target to disappear is a far easier programming task than creating the "exploding and flying out in all directions" display.

As mentioned before, most space games are simply slightly altered versions of the more conventional war games that have been played for many years on game boards. Stratego©, a very popular Parker Brothers game for many years, is a board game involving conflict and war. Here, you have captains, bombs, flags, spies, and other gaming pieces which, when arranged in a certain pattern, bring about a win. On the computer screen during a space game, the gaming pieces are video impressions, which can be manipulated via the keyboard. In other war games, if we simply rename the attacking Huns and call them the bloodthirsty Klingons, we have begun to make strides toward developing a true space game. However, the arrangement of the players and the object of the game may be identical to those which have been played for many years on cardboard. From a player's viewpoint, the space game requires no more human brain power (probably less) than the board game version, and both accomplish about the same thing. The point is, however, that all games must be designed to appeal to humans. Today, the idea of space wars has been made popular by motion pictures and television. No ones sees a straight Donald Duck cartoon anymore, but you may see on entitled *Donald Duck in Outer Space*. We are all being made aware of the mysteries and supposed excitement of conventional practices when they are attempted in a nonearthly environment. The successful game programmer must realize this and orient his games to appeal to whatever fad happens to be in vogue at the time.

Pac-Man, a currently popular space age game even has its own television series. It is one of the few video games that is not really a simulation of a real life event. I can think of no real life experience which even comes close to resembling the psychology behind the Pac-Man game. Pac-Man is popular because it is so simple that anyone can play it, from preschool age on up, and yet it requires skill and reflex actions that are a challenge to even the most intelligent individuals. It is fairly easy to come up with games that have little or no real life experience counterparts, but unless you're as inventive as the creator of Pac-Man, most of these will be popular to only a small segment of players.

SIMULATION GAMES

While it is true that all computer games are simulations of real life experiences, most which fall under the "simulated" category can be readily identified with these experiences. For example, anyone who plays a computer football game can immediately see the resemblances to the real game. In an example such as Pac-Man, it is obvious that the main character is eating something and will, in turn, be eaten himself if he does not fulfill certain obligations. While the latter game has some real life counterparts, this could not accurately be called a simulator.

Game simulations are often used for educational purposes. The competition or gaming angle helps to hold the student's attention and is really designed to make learning a pleasingly challenging process. For example, an algebra teacher may have problems getting students to concentrate on working out a particular formula because the correct answer will yield no direct result other than itself. This last statement is not entirely accurate, as the ability to work such formulas may certainly be put to use in a practical application at some later time. The computer simulator games, however, impart the element of immediacy. The student must accomplish something to bring about or prevent a certain screen or machine action. The student is competing against the machine. While I have alluded to a high school algebra class in this discussion, game simulations are used in a myriad of different areas which include the training of test pilots, to name only one. There is a popular simulation game out that goes by several different names. This is a simulation of artillery fire. A projectile is fired from a graphic cannon whose angle, and sometimes firing velocity, can be altered via the keyboard. The player sees the shell exit the barrel and then follows its flight through touchdown. Sometimes the graphics are omitted altogether, and the screen displays the distance (plus or minus) from the target of the expended projectile. This has been used as an effective tool for teaching certain branches of mathematics that relate to coordinates, trajectories, and angles. The idea is to fire the projectile so that it lands on the target, and the computer and player become automatic competitors. The student is actually playing himself, as the computer will do only what his input data tells it to do.

Stock market simulations have been developed to train new brokers, and the game data is changed periodically to reflect recent market trends. This type of simulation game allows the neophyte to be put in control of millions upon millions of dollars and to handle the investments in much the same environment as the licensed broker going about his job. When serious mistakes are made, there are no penalties, but of course, when the proper investment has been made, there are no profits either. Again, the student is playing against himself and is learning about his personal weak areas. This will carry over into the actual job when real money is involved. Such a computer simulation game may avoid the necessity of placing a promising young broker in a rather superfluous position for the first couple of years in order to keep him from making serious miscalculations. It does not entirely take the place of on the job training, but it can greatly shorten the training period, thus saving time and money for all concerned.

When simulation games are applied to human professions, there seems to be no end to what they can do and where they can be put to use. Most readers are familiar with aircraft simulators, whereby a pilot in training is actually encased in a simulated cockpit, which is as lifelike as the real thing. When the control yoke is pulled toward the pilot in training, his display screen indicates a nose-up attitude in relation to the horizon. Also, the entire module is tilted upward at the same angle that would be assumed by the aircraft in flight.

Obviously, it is not possible to accomplish all this with your microcomputer, but many flying simulations have been programmed to help train the student pilot who is trying for his private pilot's license. The cost of flying is quite high, and it's not possible to spend all of your time in a rented airplane. Besides, a large part of the private pilot's test involves course plotting, aircraft weight and balance figures, wind directions and velocities, etc. About a year ago, I wrote a computer program which caused the screen to display a reproduction of an aerial map. The student was to take off from one airport and land at another after being given a set of data which specified wind velocity and direction, aircraft fuel consumption, aircraft weight and other mandatory flight information. The student was then required to input the proper course and the estimated time of arrival. This information was figured on his flight calculator. Often, some quite outrageous conditions were specified, and unless a near-perfect course was plotted, the plane would run out of fuel before reaching its destination. The simulation actually showed a simple graphic airplane departing the home runway and flying the course the student had charted. Due to wind conditions and other factors, the plotted course and the actual course would usually be different. Sometimes, the plane would make it, and sometimes it wouldn't. Occasionally, the plane would fly completely off the screen due to a course error on the part of the student. There were varying degrees of difficulty to allow the beginner to progress at his own pace. This program was quite complex, but it served a worthwhile purpose in training local student pilots. Usually, the job of figuring out wind speed and direction, aircraft loading, and compass heading is quite boring when compared with actual plotting experience. This program, however, added the element of competition, and many students uncomplainingly put in the hours of study required. There are those who will argue that this program and many others like it are not true games, but I cannot agree.

Just because a program or any other pursuit happens to be educational does not necessarily mean that it is not a game. The purpose of a game is to provide relaxation, competition, and/or fun in general. If these specifications also result in a practical learning experience, so much the better.

Other simulation games involve training students to drive automobiles and even construct large buildings. All of them are primarily based upon real life situations and must be periodically updated to reflect changes within the field. All offer an element of competition, but most are primarily aimed at efficient education.

It is conceivable that as computers and the art of computer programming evolve, simulation programs will become more and more a part of the traditional educational system. Computer simulation programs have even been developed that simulate the operation of a large mainframe computer, but using microcomputers. The computer itself, then, becomes a training aid for the scientific field of which it is a major part. Many high schools now use microcomputers as an instrumental part of their scholastic criteria and would, by now, be lost without them.

WRITING YOUR OWN GAME PROGRAMS

By now, you have more than likely come up with ideas of your own for writing original game programs. If you're like most persons, you will probably develop a fairly simple game routine and then add to it periodically as your expertise increases. The TI-99/4A allows for much versatility in this pursuit. Expansion and modification are the keys here. You might begin by programming a popular commercial game which has been with us for many years. You might choose to change some of the rules, but more importantly, to change the game environment. An example of this was discussed earlier when the names of countries in a board war game were changed to those of planets within our solar system. The evil villain might be called a Klingon battle warrior or given some other modern designation. Using the computer, an old-time cavalry charge can be almost instantaneously converted to an attack by a squadron of star cruisers.

Start simply by choosing a favorite board or parlor game (not too difficult) and committing it to a program. Most games involve a series of simple functions, and once you've learned to program one function, the rest come quite easily. It will then be necessary, in most instances, to branch to different subroutines. After a few hours of debugging, you should end up with a close approximation of the original game. When you have arrived at this point, let your imagination take over. Some of the most outlandish ideas will make the best games. The modification of a cavalry charge game to a space game will first involve the altering of the game piece names or designations. Then comes the playing field, which might be changed from an old southern battlefield to somewhere in outer space. Through the magic of the computer, the entire scene can be shifted to the bottom of the Atlantic Ocean, using frogmen riding on portable submarines.

Again, the construction of most computer games is a step-by-step process, and the whole game is rarely worked out in the mind of the programmer before the writing process actually begins. More often, the programmer has a general idea or concept of a particular game. During the writing process, the programmer may develop many new concepts, and the original idea is altered accordingly. When the entire game is complete, the debugging process begins, and even here, new ideas, which can be easily written into the program, may develop. When the game is ready to go, it may be played and enjoyed for many hours, but the programmer will always be thinking of ideas that will bring about improvements, and this game may serve as the basis for even more complex game programs that bear little resemblance to the original.

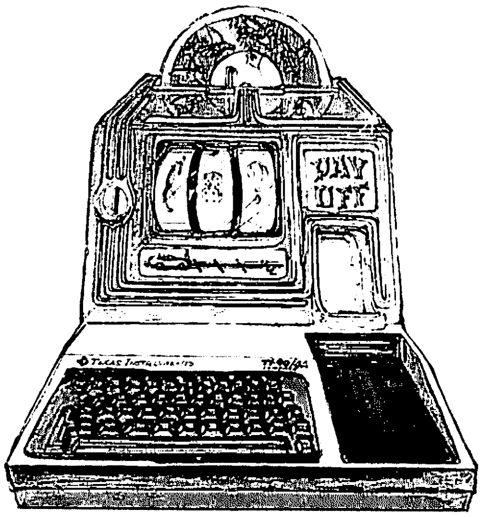
Use ideas and other games that you are quite familiar with. As a specific example, consider a simulation of the rolling of a pair of dice. A dice roll is fine for many games, but it's really out of place in a game that involves spaceships, black holes, and other celestial objects. Here, you will want to stay away from dice, but you may still need a method of generating random numbers to control game moves. If the number six can be represented by six dots on a die, couldn't it be represented as easily by six spaceships, a planet with six moons, or some other set of objects that fit well into the game format? Nor are you limited to a random number with a maximum value of six. A graphics program could display a simulation of a sophisticated instrument board and actually draw a large number six or even six million in much the same manner as the electronic digital displays that are popular in electronic equipment today. Here, each number is made up of a series of points or ASCII characters. This is much more difficult to accomplish than when using the RND function and standard-sized numbers, but if it significantly improves your game, what have you got to lose?

The TI Speech Synthesizer is something else to consider when programming computer games. This synthesizer is relatively inexpensive and is especially appropriate for space games, as its audio output closely resembles the speech patterns often associated with robots and other futuristic phenomena. It is amusing to note that the most striking contraptions seen in science fiction movies are those that closely simulate the actions of human beings. Therefore, the more human attributes you build into your program, the more your game will attract those concerned with science fiction. It might be interesting to write a game program which outputs screen prompts in a machine language that would force the human player to act more like a computer than a human being. This would put the shoe on the other foot, so to speak, but I have grave doubts about the popularity of such a game.

New computer games are being developed almost daily. Some are simply electronic versions of popular board games, while many are so modified through human imagination that they do not resemble real life experiences at all. Some are written by well-paid teams from commercial companies; others are developed by hobbyists. Some are good, and some are not so good, and this is true of games developed by both groups.

To write a successful computer game program, it is necessary to know something of psychology, especially regarding human reactions to new experiences and developments. Adventure is a big part of most computer games, and the really successful ones allow the human players to actually place themselves in the role of the gaming pieces. As trends change, so does the popularity of certain computer games. Programming the ideal game is a goal you'll never grow tired of pursuing.

Chapter 3



Graphics and Sound for Games

TI BASIC has a special set of subprograms built into the computer. These let you produce on-screen colors, graphics, and sounds. Whenever you want to use any of these special subprograms, you must call for them by name using the call statement. Additionally, you will have to provide a few specifications to be used in the subprogram. From this point on, the subprogram does the rest.

The subprograms we are primarily interested in are CHAR, VCHAR, and HCHAR. When any one of these subprograms is to be accessed during a program, you use a call statement, such as CALL VCHAR.

SCREEN COORDINATES AND ASCII

Before using the subprograms you need to understand the screen coordinates of the TI-99/4A, as well as the ASCII character set. The TI-99/4A prints characters on the screen that fill tiny blocks. Figure 3-1 shows the display screen divided into 768 blocks, each of which is the same size. The blocks are numbered horizontally, from left to right, starting at 1 and ending at 32. Blocks are also numbered vertically from 1 to 24. When discussing display screens, we refer to a horizontal line of blocks as a row and a vertical line as a column. The TI-99/4A screen consists of 24 rows and 32 columns.

Each of the 768 blocks is broken down into 64 tinier blocks, as shown in Fig. 3-2. When your screen is filled with information, 49,152 blocks have been filled in. The character set for the TI-99/4A is determined by filling in some of the 64 tiny blocks and not filling in others. Figure 3-3 shows how an O is formed on the

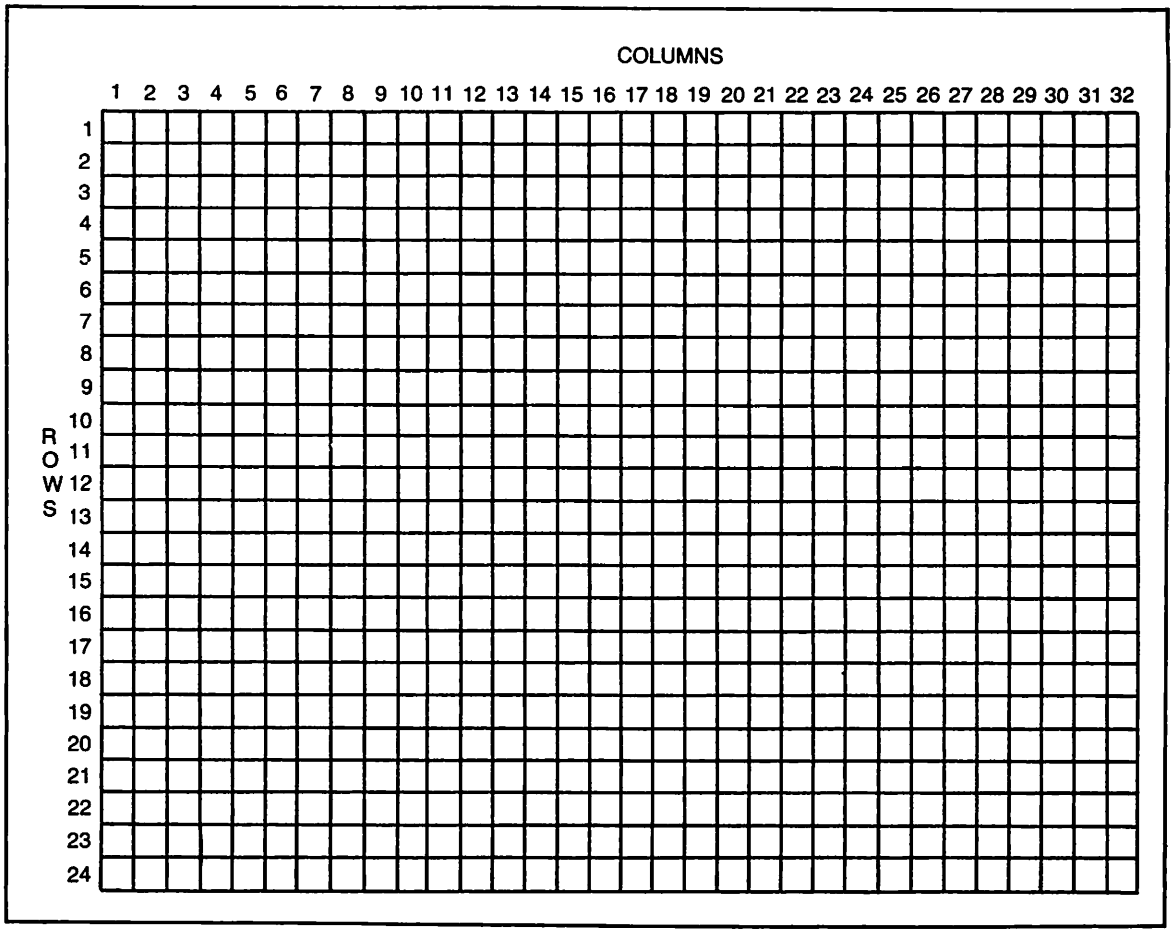


Fig. 3-1. Coordinate format of the TI-99/4A display screen.

screen by filling in some of the 64 blocks and leaving all the others vacant.

When doing on-screen graphics with the TI-99/4A, you must also understand the ASCII codes that represent the machine's character set. Each character is represented by an ASCII number. Appendix B contains the complete character set and ASCII number information and should be used as a reference whenever you're programming graphics.

A capital letter A is generated by ASCII code 65. The lowercase A is generated by ASCII code 97. A comma is ASCII code 44, and a space is ASCII code 32.

Sometimes it is necessary to print a letter, number, or character at a certain location on the screen. In this case, we cannot specify the character by its keyboard designation. We have to use its ASCII code.

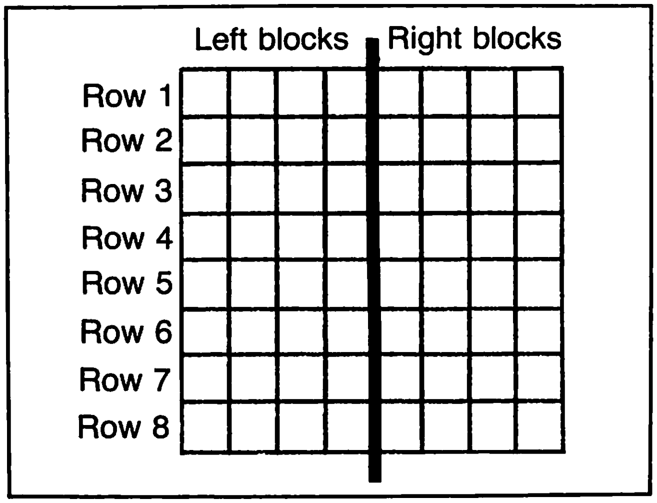


Fig. 3-2. Each character is created by filling in up to a total of 64 grid blocks.

HCHAR

The HCHAR subprogram is used to place a character anywhere on the screen by specifying the row and column coordinates. This subprogram can also repeat the character horizontally the number of times specified. This subprogram is used with the call statement, as in:

```
CALL HCHAR(12,16,65)
```

The first number in parentheses identifies the screen *row* (counting down from the top) where the character is to be printed. If you refer to Fig. 3-1, you see that row 12 is at the center left of the screen. The second character specifies the column position (counting from left to right). This lies at the center of the screen. When these two numbers are combined in this manner, you're telling the machine to print a character in row 12 at the

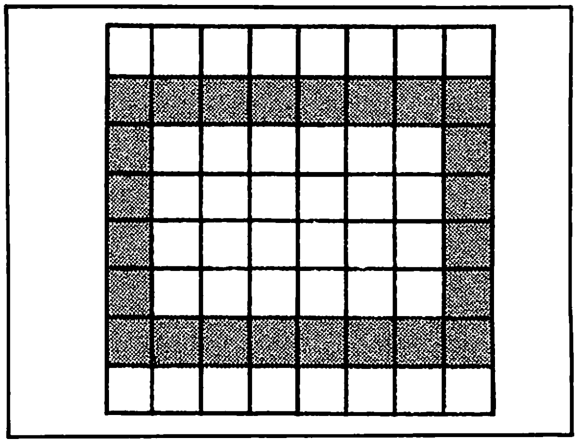


Fig. 3-3. The form of the letter O using the 64-block grid.

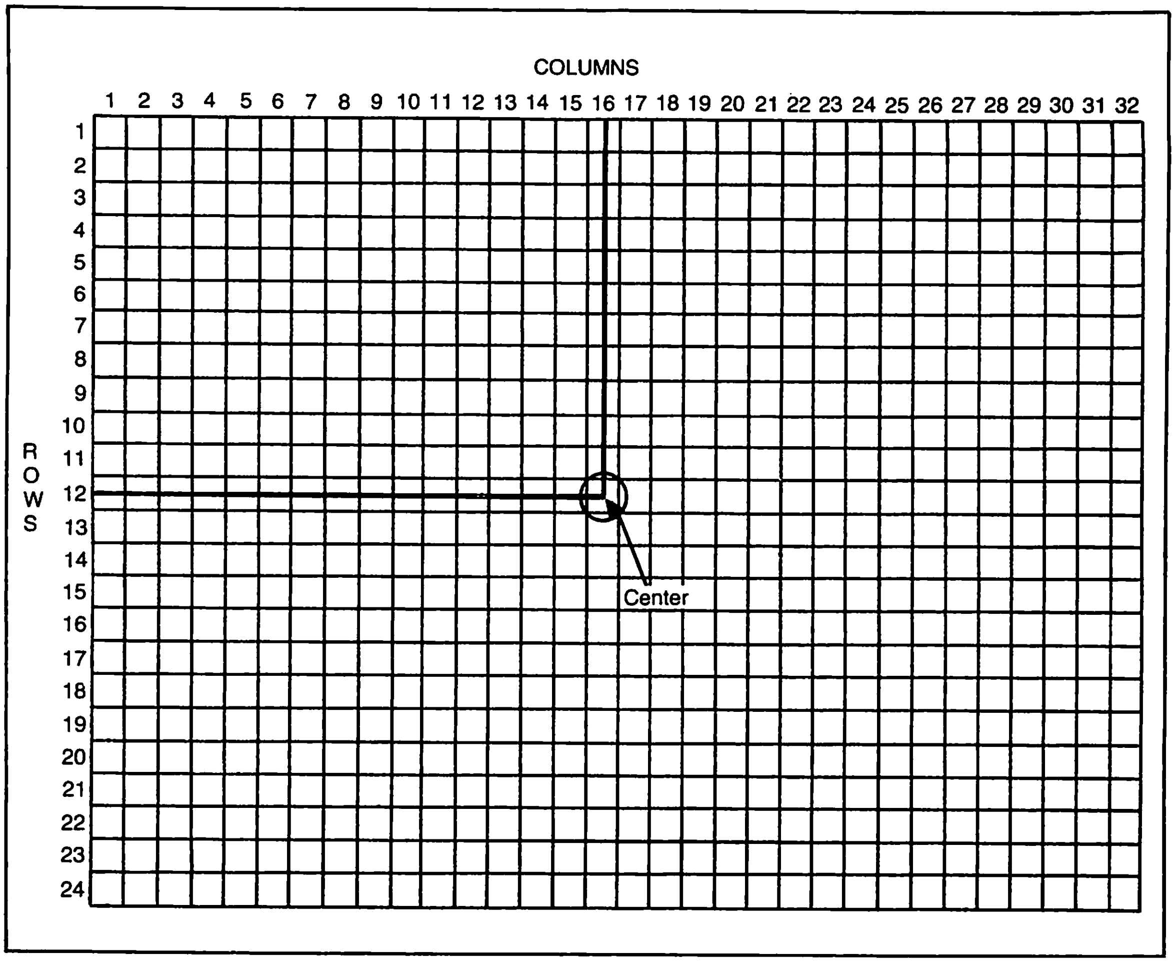


Fig. 3-4. The center of the screen is represented by the coordinate designations 12,16.

sixteenth column position. This falls in the exact center of the screen (Fig. 3-4). You first locate row 12 and then you move toward the right until you hit column 16. This is where the character will be printed.

There's a third number in parentheses, and this specifies the character to be printed, giving its ASCII code number. Here, the code is 65, so a capital letter A will be printed at screen position 12,16.

Although not shown in the previous example, you can add one more number to those already contained in the parentheses. This is called the repeat number, and it may be used to repeat the character specified any number of times. For example, in:

```
CALL HCHAR (12,16,65,5)
```

a capital letter A will be printed at the center of the screen, and as specified by the last number in parentheses (5), A will be repeated 5 times.

The first character will be printed at position 12,16. All other characters will be printed in a horizontal format to the right of the first character. The output from this program will be:

```
AAAAA
```

with the first character at screen position 12,16. The next character will be at screen position 12,17; then 12,18; etc. If you specify the repeat of 20 of these characters, as in:

```
CALL HCHAR(12,16,65,20)
```

you will run out of horizontal spaces in row 12. Remember that there are 32 columns to a single row. Since you started at column 16, this means that the row can hold only 16 more characters. By specifying the repeat of 20 characters, you run out of columns in line 12, so the machine automatically advances to row 13 and prints the additional characters there. The result will be:

```
    AAAAAAAAAAAAAAAAA
AAA
```

VCHAR

The VCHAR subprogram is identical to HCHAR, except the optional character repeat occurs in a vertical format. The following demonstrates this:

```
CALL VCHAR(12,16,65,5)
```

This program causes a vertical column of capital As to appear at the center of the screen. The first A is printed at coordinate 12,16. The second one will be at 13,16; then 14,16, etc.

If you want to print a single character at a certain location on the screen, you may use either HCHAR or VCHAR. For instance:

```
CALL HCHAR(12,16,65)
CALL VCHAR(12,16,65)
```

will produce the same results on the screen.

We can use VCHAR and HCHAR together in programs to produce simple on-screen graphic displays and even to make a chart or two. The following program makes a large letter T on the screen using small capital Ts:

```
10  CALL CLEAR
20  CALL HCHAR(6,10,84,11)
30  CALL VCHAR(7,16,84,10)
```

Line 20 causes 11 letters (T) to be printed horizontally on the screen. Line 30 causes the same letters to be printed vertically at the center of the horizontal column.

A for-next loop can be used with these subprograms to produce some interesting results, including pictures and graphs. The following program gives a simple demonstration:

```
10  CALL CLEAR
20  FOR X = 1 TO 10
30  CALL HCHAR(X,16,42,5)
40  NEXT X
```

This program produces the following results centered on the screen:

```
                    *****
                    *****
                    *****
                    *****
                    *****
                    *****
                    *****
                    *****
                    *****
                    *****
```

You can also use input statements to make an effective bar graph. The following program does this:

```
    10  CALL CLEAR
    20  INPUT A
    30  INPUT B
    40  INPUT C
    50  CALL CLEAR
    60  CALL HCHAR(5,1,42,A)
    70  CALL HCHAR(10,1,42,B)
    80  CALL HCHAR(15,1,42,C)
```

This program gives you the opportunity to enter three numeric values, A, B, and C. These values are then fed to the Call HCHAR subprograms in lines 60, 70, and 80. This generates horizontal bar graphs, starting at the left side of the screen. Line 60 specifies that the first character is printed in row 5 at column position 1. Line 70 begins the next graph by dropping down 5 rows, but again, the first character is printed at position 1. The same is true of the subprogram in line 80. Five more rows have been skipped, but the same column starting position is used. Assuming values of 8, 15, and 20 for variables A, B, and C, respectively, the following chart will be displayed on the screen:

```
    ********
    ***************
    ********************
```

Here are three bars that represent the numeric values by adjusting their lengths accordingly. The first bar contains 8 asterisks, the second 15, and the third 20.

CHAR

The Call CHAR subprogram is used when it is necessary to generate characters that are not a part of the TI BASIC character set. This subprogram lets you design your own characters by filling in the proper number of squares that make up each character block.

Figure 3-5 shows the 64 blocks that make up a single screen character block. These are broken down into eight rows, with 2 block sets per row. Each block set contains 4 squares. The first 4 blocks in a row are given a certain numerical specification, followed by a numerical specification for the second set of 4 blocks.

There are 2 numerical specifications for each row, so each block character is defined by 16 numbers (2 times 8 rows). Just remember that there are 8 rows to each character and 8 possible columns in each row. The 8 columns are broken down into 2 major sets, each containing 4 columns. Remember now, I'm speaking here of 64 tiny squares which make up one screen position.

Assume that we want to make a character that consists of filling in only one block of the 64. We have to provide a number for the one

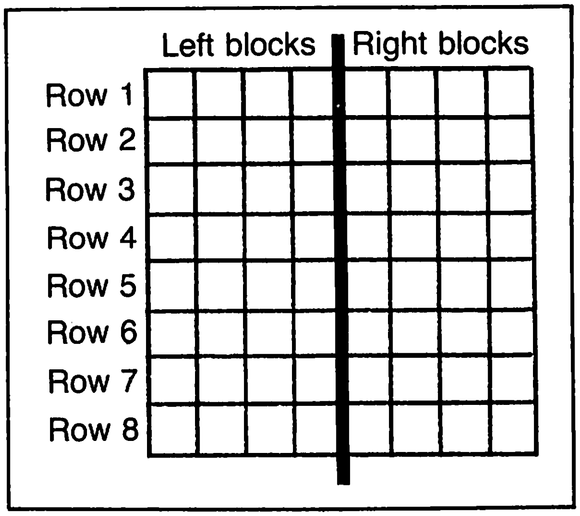


Fig. 3-5. Breakdown of the 64 blocks is handled in rows of 8.

block to be filled in, and we also have to provide numbers for those which are not to be filled in. Remember, zero is a number and is used to indicate the blocks that are not to be filled in. Figure 3-6 shows the 64-block grid with one square filled in to form a character. This square is the first one in row 1 and is specified with a certain number. The other squares are left blank, so these must be specified with another number (0).

We do not have to insert a number for each block, but rather for each set of four blocks. The first row requires two numbers to describe its two sets of four blocks. The same

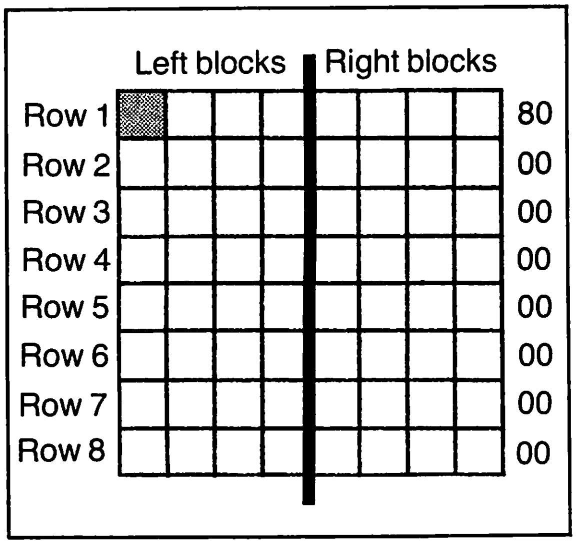


Fig. 3-6. The 64-block grid with one square filled in and each row numbered accordingly.

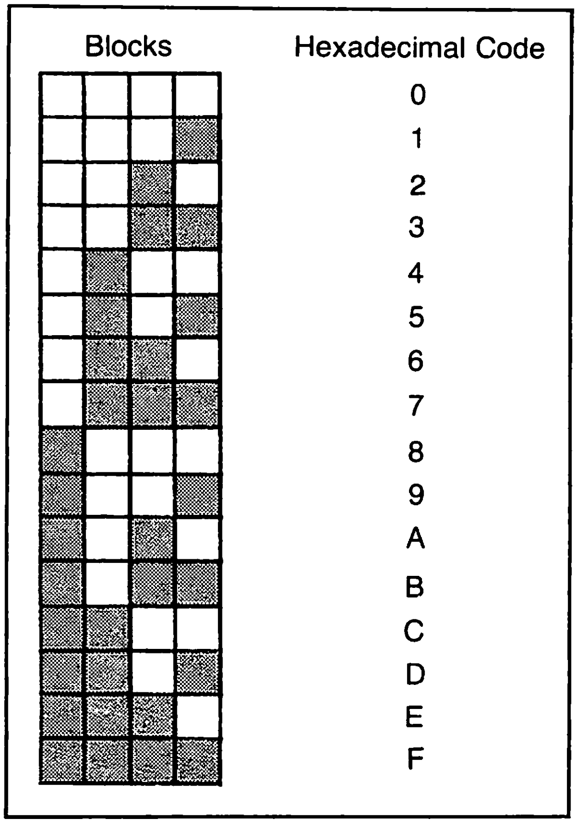


Fig. 3-7. The hexadecimal chart for filling in block grids.

applies to the remaining seven rows. Any set of four blocks uses one number to describe the blocks that are to be filled in. If the first block is to be filled in and the rest left vacant, then one number will describe this situation. If the first two blocks are to be filled in, another number will describe this. If all blocks are to be filled in, yet another number will describe this.

While there are 64 total blocks in each grid, only 16 numbers are needed to describe any possible pattern that can be derived from this grid. Two numbers are given per row.

To make things a bit more complex, the numbers that describe each block set are given in hexadecimal notation. This is just another number system using 16 as a base instead of 10, which is the base in the decimal system. It is not important to know how the system is derived or even how to convert from decimal to hexadecimal. For programming graphic characters, all you need is the chart shown in Fig. 3-7. This tells you what number or letter to use in order to describe the blocks you wish to have filled in.

Hexadecimal code uses letters to describe numbers above 10. The letter F in hexadecimal code is really a number. Looking at the chart, we see that if no blocks are to be filled in in any 4-block set, use 0. In the next row, if you wish to fill in the fourth block only, use hexadecimal code 1. This does *not* mean that if you wish to fill only one block in a row, you use the number 1. It means that if you specifically want to fill in the last block in a 4-block row, use 1. If you wish to fill in the third block, use 2; the third and fourth blocks are filled in by hexadecimal code 3; and so forth.

Figure 3-8 shows a sample pattern using the 64-block grid. This pattern was chosen at random, and the hexadecimal code for each block set of 4 is given to the right. Row 1 contains no filled-in squares. We know the hexadecimal code for no blocks filled is 0. Therefore, this row is represented by the hexadecimal code "00". The first 0 describes the left side of blocks in row 1, while the second 0 describes the right set. In row 2, the condition is the same so hexadecimal code "00" represents this row as well.

In row 3 the last block in the left block set is filled in, as is the first block in the right block set. The hexadecimal code to describe this row is "18". The 1 indicates that the last block in the left block section of row 3 is to be filled in. The 8 indicates that the first block in the right block section is to be filled in. Rows 4 through 8 follow the same pattern. In the last row, all blocks are filled in in each block set. The code that describes a completely filled in block set is F. Therefore, FF means fill in both block sets on this row.

We now have a simple character of our own design. We can display it on the screen using the Call CHAR subprogram. Our new character must be assigned an ASCII number. It can be any number on the ASCII chart used to represent a character already in the machine set. Any number from 32 to 127 will do. Let's use the character code (33) for this example. ASCII code 33 represents an exclamation point (!). However, we're going to use this number to represent our new character. The following

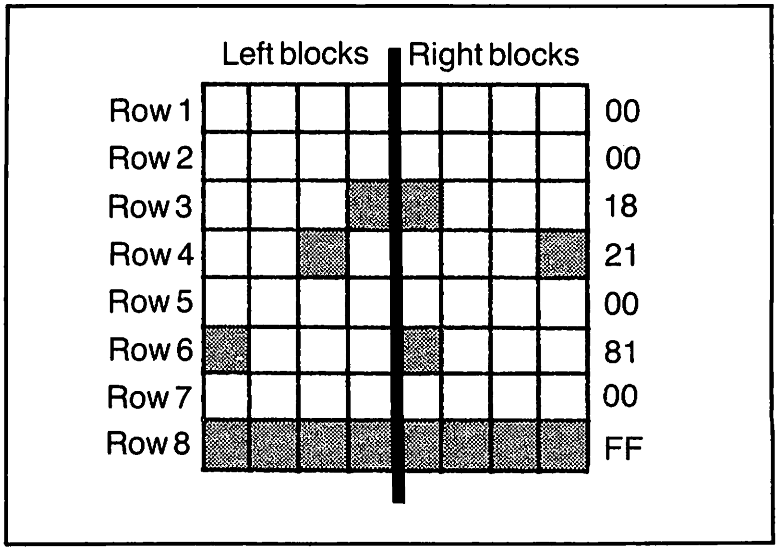


Fig. 3-8. A typical grid figure.

program defines the new character and assigns it to ASCII code 33:

```
CALL CHAR(33,"00001821008100FF")
```

This line tells the computer to assign the character identified by the hexadecimal code to ASCII code 33. The following program will display the character at the center of the screen:

```
10  CALL CLEAR
20  CALL CHAR(33,
    "00001821008100FF")
30  CALL HCHAR(12,16,33)
40  GOTO 30
```

The first line clears the screen. Line 20 then inputs the new character pattern, assigning it to ASCII code 33. Line 30 uses the Call HCHAR subprogram to locate screen position 12,16 and then print ASCII code character 33 on the screen. This would normally be "!" but since this character code has been reassigned in line 20, the pattern shown on the grid in Fig. 3-8 appears as a single character on the screen. An endless loop is set up in line 40 that continually prints the new character at the same position. Without this loop, you would see the new character, and it would then suddenly be replaced by the original ASCII character (!) as the program terminated. Unless you continually reprint your new character on the screen, it will be replaced by the character originally programmed in the character set.

Let's try some block graphics now by filling in an entire character block. The following line will do this:

```
CALL CHAR(33,"FFFFFFFFFFFFFFFF")
```

The sixteen Fs in the hexadecimal code indicate that all sixteen four-block sets (64 blocks) are to be completely filled in. This creates a solid block character on the screen. The following program prints a solid line from left to right across the center of the screen:

```
10  CALL CLEAR
20  CALL CHAR(33,
    "FFFFFFFFFFFFFFFF"
30  CALL HCHAR(12,1,33,32)
40  GOTO 30
```

After the block character has been established in line 20, the HCHAR subprogram is used to print a string of 32 character horizontally on the screen from position 12,1. ASCII character 33 is the block character established in line 20. The number 32 tells HCHAR to repeat this character 32 times, which is the maximum number of columns on any line. Your screen will display a solid line running across the screen from left to right at its center.

You could use a similar program, only substituting VCHAR for HCHAR in line 30, to draw a vertical line at the center of the screen. Here, you might include:

```
30  CALL VCHAR(1,16,33,24)
```

This would draw a vertical line starting at the top center and ending at the bottom center of the screen. By combining block character with VCHAR and HCHAR, it is possible to draw different kinds of simple pictures on the screen.

COLOR

The Color subprogram lets you change screen character colors and even the screen background. Again, the call statement is used with this subprogram. You can choose up to 16 foreground and background colors, and you can specify which set of characters will be given which color. In the TI-99/4A, there are 16 character set numbers. These are shown in Fig. 3-9. Set 1 is comprised of ASCII character codes 32 through 39. Any character represented by the numbers 32 through 39 falls into this particular set number. The set number is important, because it must be used with the Call Color subprogram to specify which characters are to be given a certain color.

Each character displayed on the monitor screen has two colors. This includes the color of the dots that make up the character itself and the color that occupies the rest of the character position on the screen. The latter are the blank spaces in the 64-square character grid. The filled-in spaces in the grid are called foreground color, while the others are background color.

Using the Call Color subprogram, you must first specify the character set number, then the foreground color, and finally, the background color. Figure 3-10 shows the 16 color codes, along with the colors they represent. If code 1 is chosen (transparent), then the present screen color shows through when the character is displayed. The following program shows how Call Color might be used:

Set Number	Character Codes
1	32-39
2	40-47
3	48-55
4	56-63
5	64-71
6	72-79
7	80-87
8	88-95
9	96-103
10	104-111
11	112-119
12	120-127
13	128-135
14	136-143
15	144-151
16	152-159

Fig. 3-9. The 16 character set numbers.

Color-code	Color
1	Transparent
2	Black
3	Medium Green
4	Light Green
5	Dark Blue
6	Light Blue
7	Dark Red
8	Cyan
9	Medium Red
10	Light Red
11	Dark Yellow
12	Light Yellow
13	Dark Green
14	Magenta
15	Gray
16	White

Fig. 3-10. The 16 color codes.

```
CALL COLOR(5,7,13)
```

This line instructs the computer to display all characters in character set 5 with a dark red foreground and a dark green background. Character set 5 includes all characters with ASCII codes of from 64 to 71. The foreground color number is 7, and this specifies dark red. The third number in parentheses is the background color. The number 13 specifies dark green.

Once the Call Color subprogram is entered, all characters represented by ASCII codes 64 to 71 will be printed on the screen in the colors previously mentioned. This particular set (5) includes capital letters A through G. If you wanted all the capital letters in the character set to be displayed on the screen in the same color, several call color statements would be necessary. All of the capital letters are included in character sets 5, 6, 7, and 8 so four Call Color subprograms would do the trick.

SCREEN

The Screen subprogram is very much like the Color subprogram, except it is used to specify the color of the screen itself. This is the palette upon which the characters are written. The same color code chart used with the Color subprogram applies to the Screen subprogram. The following program segment shows how the Screen subprogram is used with the previous Call Color subprogram:

```
10  CALL CLEAR
20  CALL SCREEN(11)
30  CALL COLOR(5,7,13)
```

This determines that the screen background color will be dark yellow (11) and that the character foreground color will be dark red with a dark green background color. Here, you have used lines 20 and 30 to control the coloration of three different screen elements: the screen itself, the character foreground, and the character background.

With these subprograms, you can highlight your displays, whether they be in alphanumeric form (text mode) or in pure graphics form. By changing the screen background colors, along with the character foreground and background colors, you can cause certain portions of a text display to be highlighted in comparison with the rest. You can also produce a myriad of multicolored images on the screen that can include kaleidoscopes and even fairly detailed pictures.

ANIMATION

The HCHAR and VCHAR subprograms can be used to produce on-screen animation, or movement.

Animation or movement is created by drawing an image on the screen in one location, erasing it, and then drawing it again at another location on the screen. If this is done rapidly enough, you don't really see the erasure process and it appears as though the object is moving instead of being written, erased, and then written again.

The program shown in Fig. 3-11 displays numbers at the center of the screen and causes them to count upward, giving the impression of motion. All that's really happening is that one number is printed; then it is written over by the next number in the sequence. This process is

```
10  CALL CLEAR
20  FOR X = 48 TO 57
30  CALL HCHAR (12,15,X)
40  NEXT X
```

Fig. 3-11. A program to display numbers at the center of the screen.

continued until the program is over. This type of procedure can be used to produce the effect of on-screen motion from left to right, bottom to top, and/or vice versa. Here's how the program in Fig. 3-11 works.

Line 10 clears the screen, and a for-next loop is entered in line 20. This causes X to count from 48 to 57 in steps of 1. These numbers represent ASCII codes in the Call HCHAR subprogram in line 30. The numbers 48 to 57 represent the ASCII codes for the numbers 0 to 9. During the first cycling of the loop, a 48 is output from the loop. This value of X is inserted into the call HCHAR statement in line 30 at the character position. Therefore, the character represented by ASCII code 48 is displayed at screen position 12,15. This character is 0. When the value of X is equal to 49 during the second cycle of the loop, a 1 will be displayed at the screen position where the 0 formerly appeared. This will continue until the loop counts to 57 and times out. The program then ends. This won't take very long, so you may wish to add another line to set up an endless loop, such as:

```
50  GOTO 20
```

This causes the program to run over and over again until manually halted.

Remember that the loop numbers 48 to 57 represent machine characters specified by ASCII character codes. Only the characters ASCII codes 48 through 57 represented appear on the screen. By changing line 20 to

```
20  FOR X = 65 TO 90
```

the capital letters A through Z appear.

In this animation program movement was confined to a single character block. It is simple to produce movement of characters from one point to another on the display screen. Let's start with the program shown in Fig. 3-12. With a few modifications this program will produce animation.

After the screen is cleared in line 10, a for-next loop is set up to count from 3 to 20 in steps of 1. Within the loop at line 30 is a Call HCHAR subprogram, which uses the value of X in the horizontal or column designator posi-

Fig. 3-12. A simple animation routine.

```
10  CALL CLEAR
20  FOR X = 3 TO 20
30  CALL HCHAR (12,X,79)
40  NEXT X
```

```
10  CALL
20  FOR X = 3 TO 20
25  CALL CLEAR
30  CALL HCHAR(12,X,79)
40  NEXT X
```

Fig. 3-13. A modification to Fig. 3-12 produces true animation.

tion. Line 30 tells the machine to print ASCII character 79 (O) at position 12,X. This means during the first cycle of the loop, the letter O will be printed at position 12,3; then at 12,4 during the next cycle, and so on, until 12,20 is reached and the loop times out. The result is 18 capital letter Os printed horizontally on the screen from position 12,3 to 12,20. All 18 appear on the screen at the same time, but you do see movement as each letter is printed in turn. You could accomplish the same thing with a single program line, such as:

```
10  CALL HCHAR(12,3,79,18)
```

This would display 18 ASCII characters identified by the number 79 horizontally on the screen starting at position 12,3.

By using a for-next loop we can set up some true animation. With one modification, a single letter (O) will travel from the left side of the screen to the right. The program is shown in Fig. 3-13. The addition is found in line 25. It's the Call Clear subprogram, which clears the screen before printing the letter O in its new position.

Here's how it works: As soon as the for-next loop is entered, the screen is cleared. The letter O is then printed at position 12,3. The loop cycles once more, and the screen is erased again by line 25. Then the letter O is printed at the next screen position (12,4). This write, erase, and write again sequence continues until the loop times out. The overall result is that of a single letter moving from left to right on the screen.

```
10  CALL CLEAR
20  FOR X = 1 TO 3Q STEP 5
30  CALL CLEAR
40  CALL HCHAR(12,X,79)
50  NEXT X
```

Fig. 3-14. Animation program.

```
10 CALL CLEAR

20 CALLCHAR(33,"FFFFFFFFFFFFFFFF")

30 FOR X = 1 TO 32

40 CALL CLEAR

50 CALL HCHAR(12,X,33)

60 NEXT X
```

Fig. 3-15. Program to animate a solid block character.

Try the program in Fig. 3-14 to make the letter travel all the way across the screen in bigger jumps.

This program is almost identical to the previous one, but the coordinates specified by the for-next loop have been modified, and the count is now in steps of 5. The first letter will be printed at position 12,1. The next will be printed at position 12,6; then 12,11, etc. The character will travel faster and in bigger jumps. You can repeat this process over and over again by adding

```
60   GOTO 20
```

This establishes an endless loop and the capital letter O will continue to race across the screen.

The program shown in Fig. 3-15 uses a trick learned earlier to cause a solid block character to race from one side of the screen to the other. Line 20 establishes the character with a Call CHAR subprogram. It assigns our new character to ASCII code 33. This character is represented by the hexadecimal code (FFFFFFFFFFFFFFFF), which fills in the character block completely. The for-next loop in line 30 is followed by a Call Clear that is also part of the loop. The next loop instruction causes our new character to be printed on the screen at various locations using Call HCHAR.

When this program is run, the block will emerge from the left side of your screen, travel to the right side, and the program will then end. This is exactly what happened with the letter O, only substituting our filled-in block character.

This left to right travel is getting rather boring, so the program shown in Fig. 3-16 reverses it. The only change is found in line 30 where the for-next loop counts from 32 to 1 instead of from 1 to 32. Loops can count up or down. However, if the starting value is more than the ending value, you must include the step command, which will be a negative number. In this case, the -1 indicates that the loop is to count from 32 to 1 in steps of -1. If we wanted to have a loop take larger steps, we might use -5. Regardless of what step is specified, it must be given as a negative number in order to count from a high number to a lower one.

```
10 CALL CLEAR
20 CALLCHAR(33,"FFFFFFFFFFFFFFFF")
30 FOR X = 32 TO 1 STEP -1
40 CALL CLEAR
50 CALL HCHAR(12,X,33)
60 NEXT X
```

Fig. 3-16. This program reverses the travel of the block.

When this program is run, the block will first appear at screen position 12,32. The next position will be 12,31, and so forth until screen position 12,1 is reached. The program then ends. The result is that instead of moving from left to right on the screen, this new program causes the block to move from right to left.

Let's combine the left to right program with the one that moves the square from right to left. The program is shown in Fig. 3-17.

```
 10  CALL CLEAR
 20  CALL CHAR(33,"FFFFFFFFFFFFFFFF")
 30  FOR X = 1 TO 32
 40  CALL CLEAR
 50  CALL HCHAR(12,X,33)
 60  NEXT X
 70  FOR X = 32 TO 1 STEP -1
 80  CALL CLEAR
 90  CALL HCHAR(12,X,33)
100  NEXT X
110  GOTO 30
```

Fig. 3-17. Program to cause block to travel left to right and then right to left.

Lines 10 through 60 are identical to the first program, and lines 70 through 100 are identical to the second program lines starting with the for-next loop. It is not necessary to redefine character 33, since this was done for both loops in line 20. Once a character is defined with a Call CHAR subprogram, the character will remain in effect whenever called for in any other part of the program. The for-next loop established in line 30 assigns X values from 1 to 32. When this loop times out, X is equal to 32. Line 70 is then executed, which establishes another loop, still using the variable X. Line 70 reassigns X from its former value to values from 32 to 1. Line 110 sets up an endless loop by branching back to line 30 after the second loop times out.

When this program is run, the block character moves from the left center of the screen to the right center. It then moves from right center to left center. This process continues until the program is manually halted.

We know that the program is really printing a multitude of characters at different positions on the screen, but erasing each old one before a new one is generated. The viewer seems to see a single cube in motion, but we know that the motion is really made up of a long series of *separate* blocks.

SOUND

The TI-99/4A has a Sound subprogram that can be used to generate a wide range of audio tones and a nice selection of audio sound effects. Most video game programs depend heavily on sound effects to make their displays and competitions more realistic.

Like the other subprograms, this one is used with the call statement. You can produce 3 simultaneous tones. Each call sound statement must include the desired duration, frequency, and volume.

Duration is given in milliseconds (1/1000 of a second) and can range from 1 to 4250. One second is equal to a 1000 milliseconds. The longest any single tone can be held is 4.25 seconds, or 4250 milliseconds.

The frequency of the tone must follow the duration command. If you want to generate tones or musical notes, the number must be anywhere from 110 to 44733 Hertz. The numbers represent frequency in Hertz (cycles per second). Tones above 44,733 Hertz (44.733 kilohertz) fall well above the human hearing range and will not be detected.

If you want to generate a noise or sound effect, specify any number from −1 to −8 for frequency. The noise produced by the TI-99/4A falls into two categories, *white noise* or *periodic noise.* You will have to test these sounds with the computer yourself. Some sound like motors running, and others offer "space" sounds, etc.

The last parameter that must be specified is volume. Volume is represented by any number from 0 to 30. Zero represents the loudest output and 30 is the softest. The following program line generates a 1000-Hertz tone for approximately 4¼ seconds at the loudest volume possible:

```
CALL SOUND(4250,1000,0)
```

The number 4250 determines the length of the tone. The value 1000 determines the frequency, and 0 determines the volume.

Figure 3-18 shows a program that generates all 8 noises or sound effects available with

```
10  FOR X = 1 TO 8

20  CALL SOUND(4250,-X,0)

30  NEXT X
```

Fig. 3-18. Program to generate all eight noises using Call Sound subprogram.

the Call Sound subprogram. The first sound generated is represented by −1, while the next sound is −2, and so on, up to −8. Each sound is held for a little over 4 seconds.

When this program is run, a for-next loop that encloses a Call Sound subprogram is entered. The value of X is from 1 to 8. These are the values of the noise numbers. The noise specification numbers must have negative values. A value of minus X is specified in the frequency section of the Call Sound subprogram contained in the for-next loop. This line tells the computer to output for 4.25 seconds the noise represented by the negative value of X. The 0 indicates that the noise is to be output at maximum volume. If the minus sign is not placed before the X, you will get an error message, because the lowest tone number that may be used is 110.

If you want to generate a multitude of tones, try the program shown in Fig. 3-19. This one is similar to the noise-generating program, but the value of X is from 110 to 2010 in steps of 100. This time, X is inserted without the minus sign, since the numbers representing tones must always be positive and equal to or more than 110. They must also be less than 44733.

When the program is executed, the first tone output will be 110 Hertz. The next tone will be 210, then 310, and so on, until a maximum frequency of 2010 is reached. I shortened the duration command in line 20 to hold each tone for about half a second, if you use a long duration command here, the program can take several minutes to run.

KEY

Another subprogram useful in maintaining control over the movement of graphic images is the Call Key subprogram. It lets you transfer one character from the keyboard directly to the program. This may sound similar to an input statement, but it's not. When an input statement is used, program execution is halted until something is input from the keyboard and the enter key is pressed. Using the Call Key subprogram, your program continues to execute in

```
10  FOR X = 110 TO 2010 STEP 100

20  CALL SOUND(500,X,0)

30  NEXT X
```

Fig. 3-19. Program to produce a multitude of musical tones.

```
 10  CALL CLEAR
 20  CALLCHAR(33,"FFFFFFFFFFFFFFFF")
 30  FOR X = 1 TO 32
 40  CALL CLEAR
 50  CALL HCHAR(2,X,33)
 60  CALL KEY(0,KEY,R)
 70  IF R = 0 THEN 90
 80  IF R = 1 THEN 110
 90  NEXT X
100  GOTO 30
110  PRINT "PROGRAM IS OVER"
120  END
```

Fig. 3-20. Use of Call Key subprogram.

a certain manner until a key is pressed. When this occurs, there is usually a branch to another portion of the program, and the program runs in a different way. It is not necessary to press the enter key after striking the key. Once you "arm" a key, the computer is constantly monitoring that key's status. As soon as the status changes (when the key is pressed), the computer reads this condition and brings about the required branch.

The Call Key subprogram is followed by several specifications in parentheses. The first is the key unit. This can be any number from 0 to 5. A key unit of 0 activates any key on the console. A key unit of 1 activates only the keys on the left side of the keyboard. These are keys 1 through 5 on the top row, Q through T on the second row, A through G on the third row, and on the fourth row, Z through B. A key unit of 2 activates the remaining keys on the right side of the keyboard. Key units 3, 4, and 5 provide specific modes for the keyboard.

Figure 3-20 shows how the Call Key subprogram might be used to provide console control during a program run. This is similar to a previous graphics program discussed in this chapter. A block character moves from left to right across the top of the screen. This is set up by the for-next loop beginning in line 30 and ending at line 90. The only thing unusual about this loop is in lines 60 through 80. In line 60, the Call Key subprogram is used to read the keyboard. R is the status factor and the third element of the Call Key subprogram. It's called the status bit, because it assumes the value or status of the keyboard. When no key is pressed, R has a value of 0. When a key is pressed, the value of R is 1. Lines 70 and 80 test for the condition of R. In line 70, if R is equal to 0, there is a branch to line 90, which simply causes the loop to cycle again. If R is equal to 1, this is detected in line 80, and there is a branch to line 110. When such a branch occurs, the moving character will freeze on the screen and the message "PROGRAM IS OVER" will be displayed.

The Call Key subprogram is often used in text mode programming in place of input

```
10  CALL CLEAR
20  PRINT "THIS IS AN INTRODUCTION TO A NEW PROGRAM CALLED MOTORCADE"
30  PRINT
40  PRINT "WHERE YOU ACTUALLY DO THE DRIVING.  THE GAME IS VERY SIMPLE TO PLAY"
50  PRINT
60  PRINT "ALONE OR WITH A FRIEND.  THE TOP ROW OF KEYS CONTROL HORIZONTAL"
70  PRINT
80  PRINT "DIRECTION.  THE SPACE BAR CONTROLS SPEED.  THE OBJECT OF THE GAME"
90  PRINT
100 PRINT "IS TO COMPLETE TEN LAPS WITHOUT STRIKING AN OBSTACLE OR ANOTHER"
110 PRINT
120 PRINT "AUTOMOBILE.  GOOD LUCK!!!"
130 PRINT
140 CALL KEY(0,KEY,R)
150 PRINT "PRESS ANY KEY TO CONTINUE"
160 IF R = 0 THEN 140
170 IF R = 1 THEN 500
```

Fig. 3-21. Using Call Key instead of Input.

statements. The program in Fig. 3-21 demonstrates this use.

This program is typical of the introductory lines of many game programs. This is only a sample used to demonstrate this use of Call Key and is not a workable program in itself. Lines 10 through 120 print game instructions on the monitor screen. As soon as the instructions are printed on the screen, nothing further occurs until the operator presses any key on the keyboard. A screen prompt appears in line 150 and tells the operator to "press any key to continue." In line 140, the Call Key subprogram is used. The 0 designator has been incorporated so that the entire keyboard is read. Line 160 brings about a branch to line 140 as long as R is equal to 0. It will be equal to this value as long as no key is pressed. This effectively sets up an endless loop within lines 140, 150, and 160. When a key is pressed, variable R will be equal to 1 and there will be a branch to another part of the program. This is detected in line 170. The branch to line 500 occurs when R is equal to 1. This fictitious line is used to represent the actual game portion of the program.

A TRUE GAME PROGRAM

The subprograms offered on the basic TI-99/4A computer combined with the statements, commands, and functions in TI BASIC give you the tools necessary to program your own video games.

To give you an example of how most of the subprograms studied in this chapter can be combined into a game, look at the Shooting Gallery program shown in Fig. 3-22. It lets you try to "blow away" a little graphic man who runs across the top of the screen. Your weapon is a graphic pen that shoots square projectiles. Each time you press any key on the keyboard, your cannon will fire. The cannon will always fire its projectile to one position on the screen. An element of skill is involved since you must fire the cannon when the running figure is at the correct position to bring about a hit.

The game includes sound effects to make it more interesting. It should only take you a few minutes to enter this program to your machine, and you can begin playing as soon as the debugging procedure is complete.

Lines 10 through 40 use REM statements to title the program and give some basic details about the memory requirements and the language used. In line 50, a call color statement is used to color the moving characters with a dark red foreground and a black background. The screen is cleared in line 60, while lines 70 and 80 develop our on-screen characters.

These are produced with call CHAR statements. In line 70, the graphic target is created from a single screen block. This will show a little man with arms extended. Line 80 prints a square block character that represents the projectile.

Line 90 brings sound effects into our program. It uses the Call Sound subprogram and causes the computer to generate a noise (−8) for approximately four seconds. When the Call Sound subprogram is used, the program lines following it are executed at the same time the sound is being heard. The sound data is fed to a buffer. This allows for simultaneous output of sound and execution of remaining program lines.

Lines 100 through 160 form a for-next loop causing the character established in line

```
10 REM SHOOTING GALLERY
20 REM COPYRIGHT FREDERICK HOLTZ AND
ASSOCIATES 1/26/83
30 REM PROGRAM RUNS IN TI-BASIC
40 REM MEMORY USED TO RUN THIS PROGRAM I
S 768 BYTES
50 CALL COLOR(1,7,2)
60 CALL CLEAR
70 CALL CHAR(33,"1818FF3C3C3C2424")
80 CALL CHAR(34,"FFFFFFFFFFFFFFFF")
90 CALL SOUND(4000,-8,0)
100 FOR X=1 TO 32
110 CALL CLEAR
120 CALL HCHAR(2,X,33)
130 CALL KEY(0,KEY,R)
140 IF R=0 THEN 160
150 IF R=1 THEN 270
160 NEXT X
170 CALL SOUND(4000,-8,0)
180 FOR X=32 TO 1 STEP -1
190 CALL KEY(0,KEY,R)
200 IF R=0 THEN 220
210 IF R=1 THEN 270
220 CALL CLEAR
230 CALL HCHAR(2,X,33)
240 NEXT X
250 CALL SOUND(4000,-8,0)
260 GOTO 100
270 FOR Y=22 TO 1 STEP -5
280 CALL CLEAR
290 CALL HCHAR(Y,15,34)
300 NEXT Y
310 IF X=15 THEN 330
320 GOTO 160
330 CALL CLEAR
340 CALL SOUND(4000,1000,0)
350 PRINT "DIRECT HIT"
360 FOR Q=1 TO 800
370 NEXT Q
380 GOTO 60
```

Fig. 3-22. Shooting Gallery program.

70 to run from left to right across the top of the screen. This is done by the HCHAR subprogram in line 120, whose horizontal coordinates are derived from the value of X. The Call Key subprogram is found in line 130, along with the test lines to bring about appropriate branches in lines 140 and 150. It takes this loop about four seconds to time out. The noise follows the little man across the screen and stops when he reaches the right side. When this occurs, the loop is timed out, and line 170 is executed. This is identical to the Sound subprogram in line 90, and sets up the sound effect for the next loop. Lines 180 through 240 cause the little man to now run from right to left across the top of the screen. This loop is identical to the previous one, except for the reverse order for the count value of X. Note the Step −1 command used at the end of line 180.

When this loop times out, there is another Call Sound subprogram (line 250) and then a branch to line 100, where the entire sequence begins again. As long as no key is pressed after the program begins running, the target will run back and forth across the screen while sound effects occur.

However, if a key is pressed, the value of R is equal to 1, and there will be a branch to line 270. Two Call Key subprograms are used in lines 130 and 190 to make sure you have control when either directional loop is executing.

When a key is pressed, the branch to line 270 causes another for-next loop to be executed. This is another negative step loop, determining the vertical coordinates for the HCHAR subprogram in line 290. The loop causes character 34 (the projectile) to move from the bottom of the screen (position 22) to the top of the screen in steps of 5 screen places.

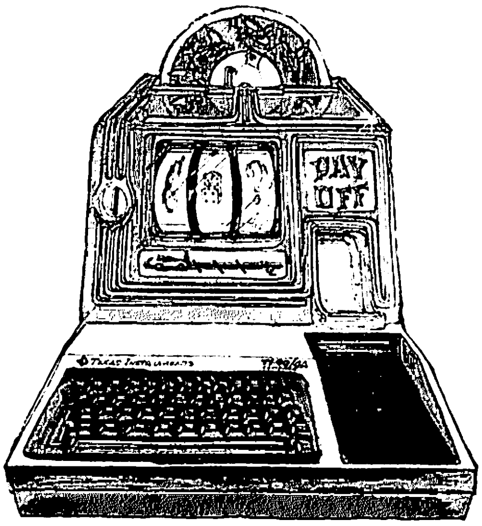
Line 310 tests for a hit. The projectile is fired from horizontal position (column) 15. It starts at 22,15, then rises to 17,15, then 12,15, etc. Line 310 states that if X is equal to 15, then branch to line 330. Here, the variable X is the last horizontal position of the little man. This was his position when the loop controlling his movement was exited by pressing a key. If the little man is in the 15th horizontal position, the projectile strikes him. When the program branches to line 330, there is a musical tone set up by the Call Sound subprogram in line 340. The screen then displays DIRECT HIT.

The for-next loop in lines 360 and 370 is a time delay loop. These two lines cause the computer to count from 1 to 800. This takes a few seconds and gives the player time to note the "DIRECT HIT" prompt on the screen. As soon as this loop times out, a branch to line 60 starts the target running again.

Going back to line 310, if X is not equal to 15, then there is a branch to line 160, which starts the program near its beginning. Each time you press a key, a single shot will be fired. It takes a bit of practice to get anywhere close to a perfect score.

This is a simple computer game program. Through a few more programming steps, a great deal more sophistication could be built in, but the program as shown is sufficient as an example of how to begin. Don't attempt to do too much too soon. The best thing to do is come up with a game idea and then program it in its simplest possible form. Once you have the basics working properly, add features to expand the enjoyment of the game. Each addition or modification should be completely debugged and made operational before you make further additions. This process may take a period of several weeks, but the end result will be a game program that is challenging, exciting, and entertaining.

Chapter 4



Game Programs

This chapter contains line listings for ready-to-run programs in TI BASIC. Each of these programs has been written and tested on the TI-99/4A home computer and has been fully debugged. Don't expect a successful run the first time through, however, as you will probably make a few keyboard errors during the inputting process. Fortunately, the excellent error-checking routines built into the TI-99 ROM will quickly identify the troublesome lines and allow you to make the needed corrections. Debugging keyboard errors usually takes less than five minutes after the program has been fully input.

Some of the game programs included in this chapter are based upon chance, while others will test your skill at beating the machine or even another player. Some are entertainment programs. Some will even help you develop a skill, such as using your computer to better advantage. You will also find a number of video action games that use on-screen animation to present an interesting display and even sharpen your hand-to-eye coordination.

All programs have been written in TI BASIC, and most are based upon a combination of straightforward routines and subroutines. A few tricks have been used here and there, but for the most part, these programs are combinations of simple short programs which intermesh to produce an interesting display and perform a myriad of operations.

You are encouraged to experiment with these programs, making changes to suit your needs or to learn more about game programming. However, I suggest that you input the programs exactly as shown in these pages be-

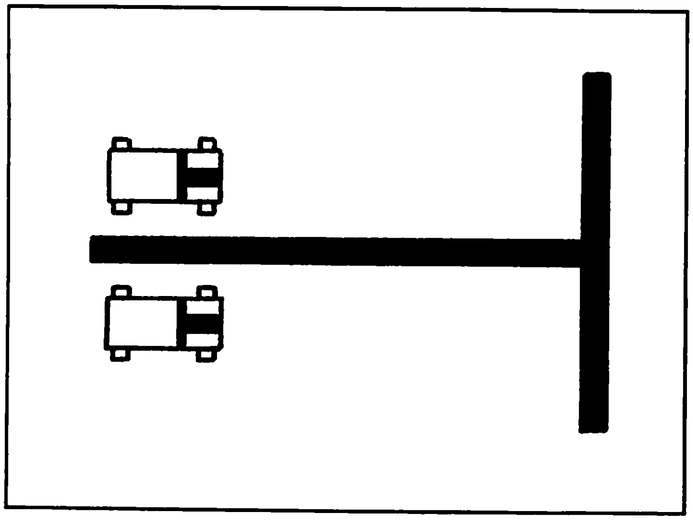


Fig. 4-1. Screen setup for auto race.

fore making any modifications. When you have a successful program run, you can then begin making changes. I suggest also that you copy the original program on cassette or disk before making any modifications. This assures an operational program from which to work, even if your modifications cause the one in current memory to be partially erased or rendered useless. The latter can sometimes occur when you make a number of changes, one or more of which may not be correct. The modified program can become unwieldy and it's sometimes easier to start from scratch than to go through and try to figure out where the problem has occurred. When this happens, you simply reload the original program from cassette or disk and start all over again.

AUTO RACE

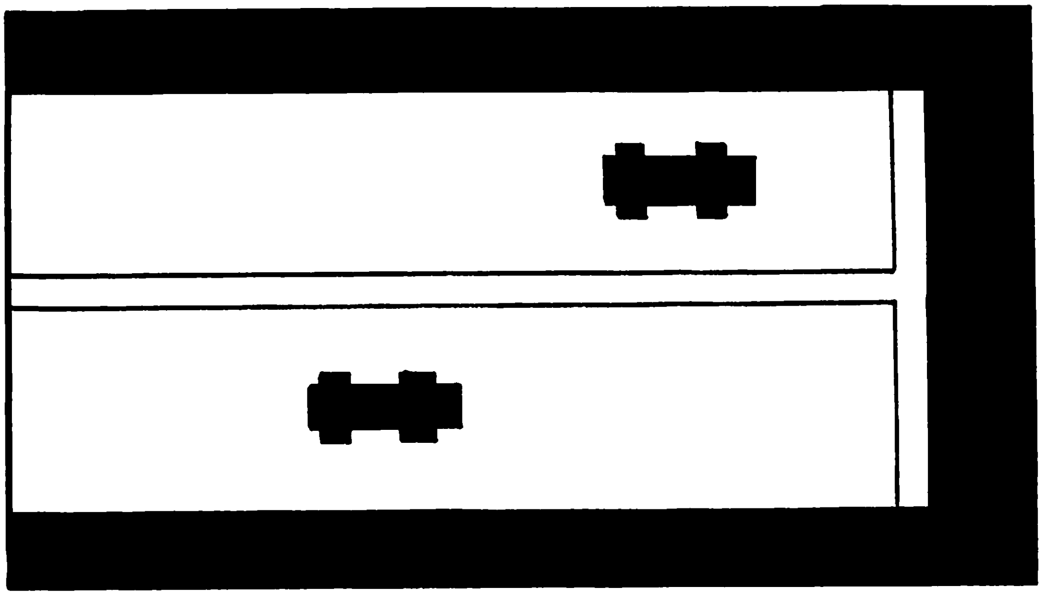
This program is a complete auto race or drag race using on-screen graphics. Two miniature automobiles streak down a graphic racetrack, speeding toward the finish line. One car may move ahead and then fall back again. You never know until the last minute just which car will win. This display uses a lot of color and will be enjoyed by young and old alike. The top car is red and the bottom one is blue. When the race is over, the screen will display the results, and you are invited to race again by pressing the enter key.

Figure 4-1 shows the screen setup. When the screen is initialized, the cars begin moving from left to right. There is always a photo-finish, in that the action freezes as soon as one car crosses the finish line. Sometimes, the race ends in a tie, and this is also recorded on the screen. Since there are two automobiles, two humans can play, each taking a different race car. This game also lends itself well to single players. Here, the human takes one car and the computer is assigned the other.

The outcome of the race is randomly decided. No player control is necessary. This may be thought of as a type of dice game or any other game whose outcome is randomly decided.

Looking at the program itself, line 130 clears any printed material from the screen. There is then a branch to line 520. This starts

Listing 1. Auto Race



```
100  REM AUTO RACE

110  REM COPYRIGHT FREDERICK HOLTZ AND ASSOCIATES 2/23/83

120  REM PROGRAM RUNS IN TI BASIC

130  CALL CLEAR

140  GOSUB 520

150  RANDOMIZE

160  CALL COLOR(1,7,2)

170  CALL COLOR(2,5,2)

180  CALL COLOR(4,11,2)

190  CALL CLEAR
```

Listing continued .

```
200  A$="FFFFFFFFFFFFFFFF"
210  B$="0000000000000000"
220  C$="0066FFFFFFFF6600"
230  CALL CHAR(60,A$)
240  CALL CHAR(34,B$)
250  CALL VCHAR(1,28,60,24)
260  CALL HCHAR(12,1,60,28)
270  CALL CHAR(33,C$)
280  CALL CHAR(40,C$)
290  FOR X = 1 TO 29
300  CALL SOUND(500,-5,0)
310  ZZ = X + INT(RND*1.5)
320  YY = X + INT(RND*1.5)
330  CALL HCHAR(5,ZZ,33)
340  CALL HCHAR(20,YY,40)
350  IF ZZ>= 27 THEN 420
360  IF YY>= 27 THEN 420
370  FOR Y = 1 TO 75
380  NEXT Y
390  CALL HCHAR(5,ZZ,34,2)
```

```
400  CALL HCHAR(20,YY,34,2)
410  NEXT X
420  IF ZZ = YY THEN 500
430  IF ZZ< YY THEN 460 ELSE 440
440  PRINT "RED CAR WINS"
450  GOTO 470
460  PRINT "BLUE CAR WINS"
470  PRINT "PRESS ENTER TO RACE AGAIN"
480  INPUT ER$
490  GOTO 190
500  PRINT "THE RACE ENDS IN A TIE"
510  GOTO 470
520  PRINT "WELCOME TO AUTO RACE"
530  PRINT "COMPUTER STYLE.  THIS IS"
540  PRINT "A GAME OF CHANCE WHICH"
550  PRINT "USES ON-SCREEN GRAPHICS"
560  PRINT "AND ANIMATION TO SIMULATE"
570  PRINT "AN AUTOMOTIVE DRAG RACE."
580  PRINT
590  PRINT "TWO DRAGSTERS WILL RACE"
```

Listing continued.

```
600  PRINT "ACROSS THE DISPLAY SCREEN."
610  PRINT "YOU AND A PLAYING PARTNER"
620  PRINT "CAN CHOOSE A RACE CAR AND"
630  PRINT "SEE WHO THE WINNER WILL BE."
640  PRINT "YOU CAN ALSO PLAY ALONE BY"
650  PRINT "CHOOSING ONE CAR AND ALLOWING"
660  PRINT "THE COMPUTER TO BE REPRE-"
670  PRINT "SENTED BY THE OTHER.  GOOD"
680  PRINT "LUCK!!!"
690  PRINT
700  PRINT
710  PRINT "PRESS ENTER TO BEGIN"
720  INPUT ER$
730  RETURN
```

the instruction sequence, which ends at line 680. The print statements are used to provide a brief explanation of what will occur when the program gets under way. Line 710 instructs the player to press the enter key in order to begin the actual race. Line 720 reads the keyboard input and the return statement in line 730 branches back to line 150, the line immediately following the GOSUB statement that initially accessed the instruction sequence subroutine.

The randomize statement is used in line 150 to reseed the random number generator. This assures race results that cannot be predicted by the human players. Lines 160 through 180 use Call Color subprograms to color the two automobiles and the racetrack itself. The call clear statement in line 190 removes the instruction information from the screen and the beginning of the graphic work is encountered in lines 200 through 280.

The first three lines in this sequence as-

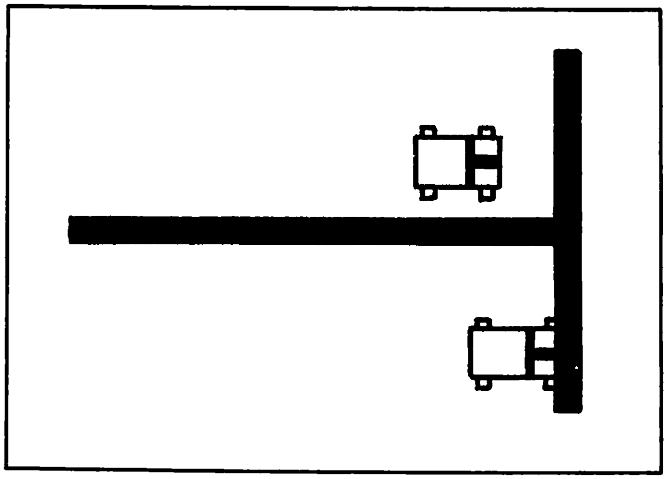


Fig. 4-2. The racing car image.

sign hexadecimal values to string variables A$, B$, and C$. The first uses sixteen Fs. This produces a solid block character when assigned by the Call CHAR subprogram in line 230. Line 210 produces a null block character or one that is the same color as the screen background. This one is assigned to ASCII character 34 by the Call CHAR subprogram in line 240. Line 220 assigns to C$ a hexadecimal value that will draw a race car on the screen. This is shown in enlarged form in Fig. 4-2. Lines 270 and 280 assign this block value to ASCII characters 33 and 40. Two assignments must be made because each car is to have a different color. ASCII character 33 lies in character set 1, while ASCII character 40 is in set 2. The Call Color subprogram in line 160 assigns the color red (7) to all characters in set 1. The Call Color subprogram in line 170 assigns the color blue (5) to all characters in set 2.

Lines 250 and 260 are used to draw the racetrack separator and the finish line on the screen. Line 250 draws the vertical line using VCHAR and ASCII character 60, which is the block character assigned in line 230. The first block character is printed at position 1,28 on the screen, and 23 more will be printed vertically downward from this point as indicated by the last numeral in the VCHAR command.

Line 260 draws the horizontal divider that uses the same block character, but begins at the left center of the screen (12,1). 28 of these characters are printed horizontally, the last intersecting with the center of the vertical finish line.

The actual race sequence can now begin, and this is handled by a for-next loop which starts in line 290. The value of X is set from 1 to 29 in increments of 1. These values will be used directly as column coordinates to allow each race car to move across the screen. Some audio effects are created by the Call Sound subprogram in line 300. I elected to use a −5 noise designator here. You may wish to use another. Lines 310 and 320 assign coordinate values to variables ZZ and YY based upon the value of X (determined by the loop value) added to the integer output of the random number (RND function) multiplied by 1.5. In some cases, this integer value will be equal to 1, and in others, it will be equal to 0. This determines the movement of the car across the screen. Each time the loop cycles, X increases by 1, but ZZ or YY may increase by 2 depending on the value of RND*1.5. Lines 330 and 340 use Call HCHAR subprograms to write the graphic automobiles at vertical screen positions 5 and 20, respectively. The horizontal position of each dragster is determined by the value of ZZ and YY. Lines 350 and 360 test for a win, which occurs when either ZZ or YY is equal to or greater than 27. When this occurs, there is a branch to line 420, which will be discussed shortly. Assuming that the cars have not progressed this far yet, line 370 is executed. This starts a nested loop, which is used for time delay purposes. As soon as the dragsters are drawn on the screen, this loop delays further execution for about a second, allowing the cars to be clearly seen by the players. When the loop times out, lines 390 and 400 are encountered. These write two more characters on the screen at the same locations occupied by the graphic automobiles. ASCII character 34 is used here, and this was assigned in line 240 to a null character or one which is the same color as the screen background. When the two characters specified by lines 390 and 400 are written on the screen, the graphic automobiles are effectively erased.

Line 410 causes the loop to cycle again, and the same sequence of events reoccurs. However, the value of X has increased by 1, and this causes the new graphic automobiles to be written at least one position farther down the track (horizontally). The actual position will be determined by the value of ZZ or YY.

When one or both cars make it to the finish line, program lines 350 or 360 sense this condition and bring about a branch to line 420. Lines 420 through 460 determine which car will be announced the winner. For instance, in line 420, a test is made for a tie. If the value of ZZ (the top car position) is equal to YY (the position of the bottom car), there is a branch to line 500, which displays the TIE announcement on the screen. In line 430, a test is made to see if ZZ is less than YY, and if so, the bottom car (blue) wins. If it is not, ZZ must be greater than YY, and the red car is the winner.

After the winner (or the tie) is determined and pictorially announced, line 480 temporarily halts program execution until the enter key is pressed. When this occurs, there is a branch to line 190, which clears the screen, and the program begins anew.

This is certainly not a program of skill, so you won't be given a test of your mental powers or even your reflexes. It is one, though, that can be enjoyed over and over again and can be played by persons of all ages.

Many persons ask why only two automobiles are included rather than four or more. After all, this would change the odds considerably and even allow for more players to take part. Certainly, more automobiles could be included, but probably at the expense of game enjoyment, at least from a visual point of view. The on-screen motion of the two automobiles is accomplished by writing the images on the screen (one at a time) and then erasing them (again, one at a time). All of this takes a short amount of execution time and even then, the motion of the automobiles does not appear to be exactly continuous, as you can actually see them being quickly written, erased, and then written again. If you added two more automobiles to this display, the execution time within the loop would be nearly doubled. This would increase the flicker effect and the speed with which the automobiles travel from the left to the right side of the screen would be slowed. The machine is simply not fast enough to accomplish this type of animation with reasonable visual efficiency, at least when using BASIC. The same can be said of most other home computers. When animation is to be produced in a highly efficient and professional manner, the BASIC language is rarely used. Such programs are normally written in assembler or machine language. Such languages are not within the scope of this book, but are certainly available to those programmers who want to go further on the TI-99/4A.

SPEEDREAD/TYPE

Speedread/Type is a learning game that will allow you to sharpen your speedreading skills and practice typing on the TI-99/4A at the same time. When the program is first run, a single line of words and/or numbers is displayed on the screen for a second or so. As soon as the phrase disappears, you must duplicate it by typing in the same information via the keyboard. The keyboard information must be identical to what appeared on the screen or a wrong answer is recorded. At the end of the game, the number of incorrect phrases is registered as your score. The individual with the lowest score is the winner.

Listing 2. Speedread/Type

SPEEDREAD

```
100  REM SPEEDREAD/TYPE

110  REM COPYRIGHT FREDERICK HOLTZ  AND ASSOCIATES 2/22/83

120  REM PROGRAM RUNS IN TI BASIC

130  CALL CLEAR

140  PRINT "THIS GAME WILL ALLOW YOU TO"

150  PRINT "PRACTICE TYPING WHILE YOU"

160  PRINT "LEARN TO READ AND COMPREHEND"

170  PRINT "SCREEN INFORMATION QUICKLY."
```

Listing continued.

```
180  PRINT "THE SCREEN WILL DISPLAY A"
190  PRINT "PHRASE WHICH WILL DISAPPEAR"
200  PRINT "AFTER A FEW SECONDS.  FROM"
210  PRINT "MEMORY, YOU MUST THEN RETYPE"
220  PRINT "THE PHRASE EXACTLY AS IT"
230  PRINT "APPEARED.  AT THE END OF THIS"
240  PRINT "GAME, YOUR SCORE WILL BE"
250  PRINT "COMPUTED.  GOOD LUCK!!!"
260  PRINT
270  PRINT
280  PRINT "PRESS ENTER TO CONTINUE."
290  INPUT ER$
300  CALL CLEAR
310  READ A$
320  IF A$="END" THEN 930
330  PRINT  A$
340  PRINT
350  PRINT
360  PRINT
370  PRINT
```

```
380  PRINT
390  PRINT
400  PRINT
410  PRINT
420  PRINT
430  FOR X=0 TO 250
440  NEXT X
450  CALL CLEAR
460  PRINT "RETYPE THE PHRASE"
470  PRINT
480  PRINT
490  INPUT AA$
500  IF AA$=A$ THEN 510 ELSE 630
510  PRINT
520  PRINT
530  PRINT
540  PRINT "THAT IS A CORRECT ANSWER"
550  PRINT
560  PRINT
570  PRINT
580  PRINT
```

Listing continued.

```
590  S=S+1

600  PRINT "PRESS ENTER TO CONTINUE"

610  INPUT ER$

620  GOTO 300

630  PRINT "THAT IS AN INCORRECT PHRASE!"

640  PRINT

650  PRINT

660  PRINT

670  PRINT "THE CORRECT PHRASE IS-"

680  PRINT

690  PRINT

700 PRINT A$

710  PRINT

720  PRINT

730  T=T+1

740  PRINT "PRESS ENTER TO CONTINUE"

750  INPUT ER$

760  GOTO 300

770  DATA DOWN BY THE OL' MILL RUN, SEVEN SONS FOR SEVEN
     DARTERS, WHEN IN ROME DUE AS THE ROAMIN'S DO

780  DATA IT WAS A VERY GOOD TEAR
```

```
790  DATA STROLLING DOWN THE MAIN

800  DATA STAUNCH IS AS  STAUNCH DOES

810  DATA COMPUTER ARITHMETIC IS FRAUGHT WITH LOGIC

820  DATA HE WAS BOOLEANED OF THE STAGE

830  DATA THE COMING OF THE HESPERUS WAS A SIGHT TO BEHOLD.

840  DATA YOU DID NOT SAY I COULD GO!

850  DATA FRONT THE ROYAL OAK!!!

860  DATA 9675892165;

870  DATA DOWN IN THE VALLEY OF THE FERAL DOGS.

880  DATA IS THIS THE CORRECT ADDRESS OR IS IT ELSEWHERE?

890  DATA LATCH HATCH SATCHEL MATCH

900  DATA IN THE FIRST OF THE NINTH THERE WERE NO RUNNERS ON BASE!!!

910  DATA VERY VENTURESOME VAGABONDS VEERED VOICELESSLY

920  DATA END

930  CALL CLEAR

940  PRINT "CORRECT PHRASES=";S

950  PRINT

960  PRINT

970  PRINT

980  PRINT "INCORRECT PHRASES=";T

990  END
```

Looking at the program, you can see that lines 100 through 280 initialize the screen, print instructions, and allow you to continue by pressing the enter key. When this is done, line 300 clears the screen, and a read statement is encountered in line 310. The read statement pulls information from a data statement. A group of these are used to provide the on-screen phrases. The data statements begin in line 770. Line 320 tests for the last data statement. This is found in line 920, and its only item is the word END. When A$ in line 320 is equal to this word, there is a branch to line 930, which clears the screen and prints the number of correctly typed phrases. Line 980 prints the number of incorrect phrases. Lines 340 through 420 contain print statements. You will notice that no variables follow these statements, nor are there words or phrases contained in quotation marks. These print statements are used solely to position the phrase to be retyped at the center of the screen. The actual phrase is typed at the bottom of the screen in line 330. The succeeding print statements simply act like carriage returns, and the displayed phrase is advanced upward to the center of the screen.

Lines 430 and 440 provide the time delay loop that determines the amount of time allowed for the message to be displayed after it has reached its resting position at the center of the screen. These lines cause the variable X to count from 0 to 250, which provides a display time of about one second. As soon as the loop times out, line 450 clears the screen, and you are then prompted to retype the phrase you've just seen.

Your input phrase is assigned to the variable AA$ in line 490. Line 500 tests to determine whether or not AA$ is equal to A$. The latter variable represents the phrase originally displayed. If the two are equal, there is a branch to line 510, which causes the phrase, THAT IS A CORRECT ANSWER to be displayed at the center of the screen. Again, print statements are used to position this phrase. Line 590 counts the number of correct answers. The value of S steps in increments of one each time the correct answer portion of the program is accessed. Following this sequence, you are prompted to press the enter key to continue, and there is a a branch to line 300, which displays another data item phrase.

Going back to line 500 in the program, if AA$ and A$ are not identical, there is a branch to line 630, which causes the INCORRECT PHRASE message to appear on the screen. After a few more print statements (for spacing purposes), line 670 is accessed and reprints the original phrase. Line 730 contains another counting routine, which assigns to the variable T the number of the incorrect phrases input. Pressing the enter key brings about a branch to line 300, and the next data item is accessed.

Lines 770 through 920 contain the data statements and their various items. This is where you can customize the program to suit your own tastes. The data statements may contain any information you desire, but remember that you can use commas only to separate data items. If a comma appears in a quoted phrase, the portion of the phrase to the left of the comma will be displayed during one program run, and the data to the right will be displayed as a separate data item during the next run.

There are other ways to custom tailor this program as well. For example, you can change the count value of X in line 430 to display the phrase contained in the data statement for a longer or shorter period of time. By increasing the value, the phrase will be displayed for a longer period of time. Decreasing it also decreases display time, since the loop times out in a shorter period.

Print statements were used to locate the phrases and certain screen prompts at a particular vertical position. This could also have been done using Call HCHAR or Call VCHAR subprograms, but then it would have been necessary to include a separate program line for each letter in the phrase and also to convert alphabetic and numeric characters into their ASCII equivalents.

For example, to print the word HELLO near the left center of the screen, the following program lines would be required:

```
100   CALL HCHAR(15,1,72)
110   CALL HCHAR(15,2,69)
120   CALL HCHAR(15,3,76)
130   CALL HCHAR(15,4,76)
140   CALL HCHAR(15,5,79)
```

The final number in each program line represents the appropriate letter of the alphabet needed to spell the word HELLO. Line 120 could be changed to:

```
120   CALL HCHAR(15,3,76,2)
```

and line 130 could be deleted, since lines 120 and 130 have both been used to produce the letter L at certain positions on the screen. This method will work only when identical letters are side by side in any given word. One can see that this method is rather tedious, so it's far simpler to print the word HELLO using a print statement, as in:

```
100   PRINT "HELLO"
```

and then follow this with a series of print statements used as spacers. This has been done throughout this program. It greatly shortens input time and produces the same results on the screen.

MONSTER

Monster is a game that revolves around random occurrences just like a dice game, except that instead of 6 possible results as in the case with a single die, there are 9 possibilities. This game uses on-screen graphics to draw a series of 9 boxes horizontally on the screen. Each one of these boxes is represented as a door, and above each door, there is a number from 1 to 9. You open the door of your choice by typing its number and then pressing the enter key. The door will then disappear if there's nothing behind it, but if this is the door the monster has been hiding behind, it will spring

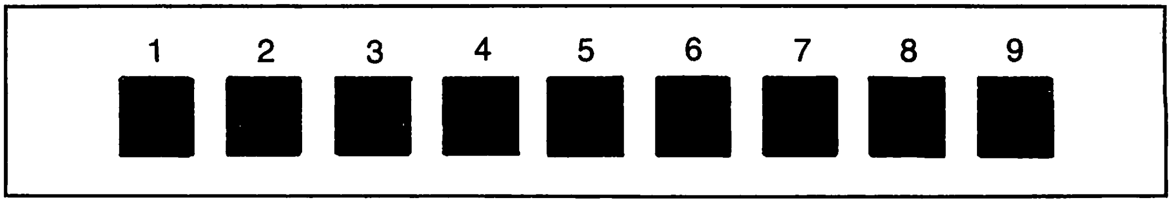


Fig. 4-3. The playing screen for monster.

out at you with a growl. Figure 4-3 shows the screen when play begins. Each time you select a door (without getting the monster), the door will disappear, but the identifying number overhead will remain. When the door with the monster is selected, the entire screen is cleared, and the monster, shown in Fig. 4-4, appears. At the same time, the excellent audio qualities of the TI-99/4A are utilized to produce a low-frequency noise that sounds like a growl.

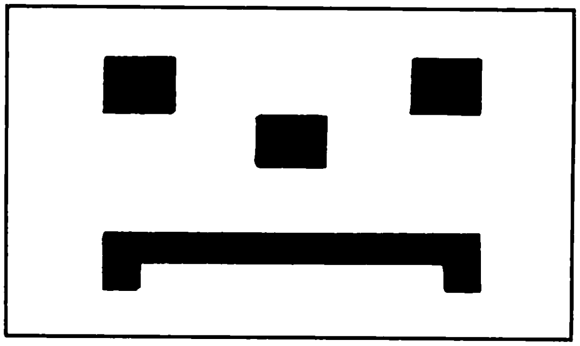
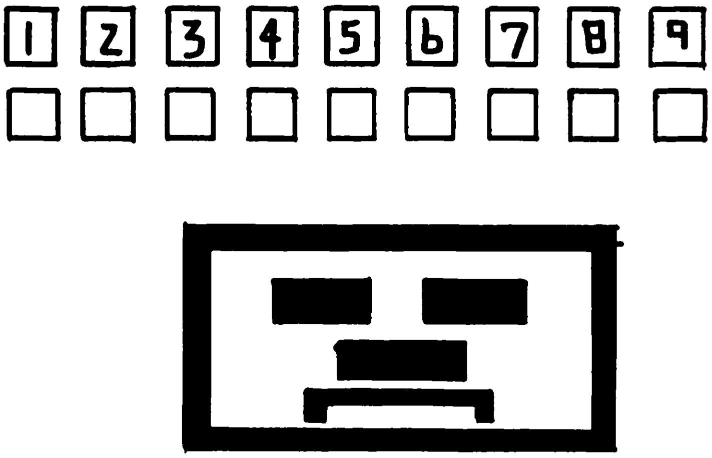


Fig. 4-4. Screen display when monster appears.

Listing 3. Monster



```
100  REM MONSTER

110  REM COPYRIGHT  FREDERICK HOLTZ  AND ASSOCIATES 2/23/83

120  REM PROGRAM RUNS IN TI BASIC

130  CALL CLEAR

140  GOSUB 690
```

```
150  DIM A(100)
160  I=0
170  RANDOMIZE
180  CALL CLEAR
190  CALL COLOR(1,3,7)
200  RN = INT(RND*9) + 1
210  A$ = "FFFFFFFFFFFFFFFF"
220  B$ = "0000000000000000"
230  CALL CHAR(33,A$)
240  CALL CHAR(34,B$)
250  CALL CLEAR
260  FOR X = 8 TO 24 STEP 2
270  CALL HCHAR(8,X,33)
280  FOR II = 1 TO 10
290  IF A(II)  = 0 THEN 320
300  CALL HCHAR(8,A(II)*2+6,34)
310  NEXT II
320  NEXT X
330  FOR Y = 8 TO 24 STEP 2
340  CALL HCHAR(6,Y,(Y/2)+45)
350  NEXT Y
```

Listing continued .

```
360  I=I+1
370  INPUT "WHICH NUMBER?":A(I)
380  IF A(I)< = 0 THEN 370
390  IF A(I)> 9 THEN 370
400  IF A(I) = RN THEN 420
410  GOTO 250
420  CALL CLEAR
430  CALL COLOR(1,5,2)
440  CALL SOUND(4200,-3,0)
450  CALL HCHAR(8,8,33,16)
460  CALL VCHAR(8,8,33,12)
470  CALL HCHAR(20,8,33,17)
480  CALL VCHAR(8,24,33,12)
490  CALL HCHAR(12,12,33,2)
500  CALL HCHAR(12,20,33,2)
510  CALL VCHAR(13,16,33,2)
520  CALL VCHAR(13,17,33,2)
530  CALL HCHAR(17,12,33,9)
540  CALL VCHAR(17,12,33,2)
550  CALL VCHAR(17,21,33,2)
```

```
560  FOR II = 1 TO 100
570  A(II) = 0
580  NEXT II
590  FOR TD = 0 TO 1200
600  NEXT TD
610  CALL CLEAR
620  I = INT(I/9*100)
630  IF I<15 THEN 640 ELSE 650
640  I = 0
650  PRINT "YOUR SCORE IS ";I
660  I = 0
670  INPUT "PRESS ENTER TO PLAY AGAIN":A$
680  GOTO 150
690  PRINT "WELCOME TO THE GAME CALLED"
700  PRINT "MONSTER.  THIS IS A GAME"
710  PRINT "OF CHANCE IN WHICH YOU"
720  PRINT "MUST LOOK BEHIND DOORS,"
730  PRINT "HOPING THE MONSTER IS NOT"
740  PRINT "THERE.  SIMPLY TYPE IN THE"
750  PRINT "NUMBER OF THE DOOR YOU"
760  PRINT "WISH TO OPEN.  YOU HAVE"
```

Listing continued.

```
770  PRINT "A ONE IN NINE CHANCE OF"
780  PRINT "OPENING THE ONE WITH THE"
790  PRINT "MONSTER BEHIND IT LAST."
800  PRINT "WHEN THIS OCCURS, YOU HAVE"
810  PRINT "BEATEN THE COMPUTER!"
820  PRINT "GOOD LUCK!!!"
830  PRINT
840  PRINT
850  PRINT "PRESS ENTER TO PLAY"
860  INPUT A$
870  RETURN
```

When the program is first run, the screen is cleared in line 130, and line 140 branches to near the end of the program, where a set of instructions is provided. Lines 690 through 870 print the instructions on the screen and allow you to press the enter key to begin program play. When enter is pressed, line 870 branches back to line 150, which follows the GOSUB statement.

In line 150, an array which will contain 100 elements is established. The array is given the designation "A". Line 170 contains a randomize statement, which assures a non-predictable random number output. The screen is then cleared again (line 180). It is initialized with a Call Color subprogram in line 190. This produces a screen that has a medium green foreground and a dark red background. This will be the color combination of the doors.

Line 200 assigns a random value between 1 and 9 to the variable RN. This is the number that determines which door the monster is hiding behind. Here, RN is assigned the integer value of RND*9 + 1. This means the number will always be an integer between and including the numbers 1 and 9.

Line 210 assigns a hexadecimal value to string variable A$. This is the value which will produce a solid block character when coupled with a Call CHAR subprogram. Line 220 also makes a hexadecimal assignment, this time to string variable B$. The 16-digit number produces a null character or a blank space on the screen. Lines 230 and 240 utilize A$ and B$ in Call CHAR subprograms. The block character is assigned to ASCII character 33, whereas the null or blank character is assigned to ASCII character 34.

Lines 260 through 320 comprise a nested for-next loop. The major loop is begun in line 260 and allows X to count from 8 to 24 in steps of 2. Line 270 feeds the value of X into the Call HCHAR subprogram. Here, the value of X is used as a horizontal screen coordinate. Notice that ASCII character 33 is specified at the end of the Call HCHAR subprogram in line 270. This means that the block character defined in line 210 and assigned in line 230 will appear in the eighth row and at horizontal position 8,10,12,14, etc. Again, the horizontal position is determined by the value of X in line 260.

The nested loop is begun in line 280. This assigns a value from 1 to 10 to numeric variable II. It's difficult to understand this phase of the program without skipping to another portion. You will remember that variable A represents the array which was established in line 150. At present, the array has no elements; therefore, its value is 0. This condition is true only upon first running the program and before any keyboard information is input. Let's skip to line 370, which assigns a keyboard value to the array. The previous for-next loops have been used to write the doors on the screen and the numbers as well. In line 370, you are asked to input a numerical value corresponding to the door you wish to open. When this is done, the array is assigned this value at one element position. Line 360 is a counting routine that steps the element positions each time line 370 is accessed. Lines 380 through 410 check for a match between your input value and RN and also for an illegal input number. For example, if you input a value which is lower than 0, as in −1, −2, etc., this is detected in line 380, and there is a branch to line 370, which asks you to input the number again. Line 390 checks for a value which is higher than 9. Line 400 checks for a value which is equal to RN. You will remember that this variable represents the number which causes the monster to appear on the screen. If this is true, there is a branch to line 420, which includes the lines for the monster image.

If, however, your number is correct (between 1 and 9) and is not equal to RN, line 410 branches to line 250. The screen is cleared, and the doors are printed one more time with one exception. The door which you selected will disappear. This is handled in a rather unusual fashion, so follow closely.

First, line 270 causes the first door to be printed at position 8,8 on the screen. Let's assume that you chose door 1 during the first go-round and that this door did not contain the monster. All right, we have our first door on the screen, but the loop is far from timing out. Line 300 calls up the blank graphic character which was assigned in line 240. It will print this at position 8,A(II)*2+6,34. Here is where the array comes into play. A(II) is equal to the first character input via the keyboard. In this case, it is 1, so a blank character will be printed at vertical position 8 and horizontal position (1*2+6), or horizontal position 8. This happens to be the same position occupied by the first block character or door. Therefore, that door is erased, or more accurately, replaced with a null character. Either way, you can no longer see it. Line 310 causes the loop to recycle, and the next door is printed. Line 300 is accessed again, and this time, a null character is printed again. However, here, the second element in array A is equal to 0, so the null character is printed at position 8,0*2+6. Zero times two is zero, so the null character is printed at position 8,6. This position on the screen is already blank, so you don't even know it's being printed. The loop will continue to cycle until all nine doors have been printed on the screen, but remember that the first door (in this case) has been erased, so only eight remain in the player's view on the screen.

Let's go around one more time and assume that the next character input is the number 4. Again, this is the second go-around, so the numbers selected are 1 and 4. The same routine occurs assuming that the number 4 does not pick up the monster. The screen is cleared, and the first door is printed in line 280 and then just as quickly erased in line 300. However, the nested loop is repeated ten times before the next door is printed (in the X loop). During the second cycle of the nested loop (II), A(II) will be equal to 4, as this was the second number the player input in line 370. Therefore, the null character is printed at horizontal position 4*2+6, or 14. This is the screen position that will be occupied by door number 4 . . . . but this door hasn't been printed yet. Be patient and you'll understand in a minute. The nested loop cycles again and from here on out, the value of A(II) will be zero. Therefore, the null character is printed at position 8,6, which has no bearing on the screen display. When the nested loop finally times out, line 320 returns to line 260 for the second cycle of the original loop. Here, the second door is printed and the nested loop goes through its gyrations again. The same thing occurs on the third cycle of the major loop, where the third door is printed. Therefore, three doors have been printed on the screen. The first one has been erased, and we are now in the fourth cycle of the major loop. Here, the fourth door is printed, but when the nested loop is entered, line 300 once again prints that null character. But this time, the effect is noticeable, as this is the position that door 4 occupies. Door 4 is printed and then immediately erased. When the X loop is finally timed out, there will be seven doors on the screen. Doors 1 and 4 will not be seen.

To summarize, during each loop cycle, the X loop repeats itself nine times, but the nested loop (II) repeats itself ten times for each cycle of the X loop. Therefore, when the X loop has timed out, the II loop has cycled ninety times. In each case, nine doors are always printed, but any doors you have previously opened are just as quickly erased from the screen. The array is necessary to keep track of your selections and to feed them back into the program during each cycle.

During each graphic cycle, lines 330 through 350 are used to place numbers atop the doors. This is done in a slightly unusual fashion too, since it is necessary to use ASCII character numbers. First, loop Y in line 330 counts in exactly the same fashion as loop X in line 260. Some mathematics are involved, however, in getting the numbers to be displayed correctly. The numbers themselves are represented by the formula (Y/2)+45. The unaltered value of Y is used to determine the horizontal print position. When Y is equal to 8, Y/2+45 is equal to 49. ASCII character 49 is the number 1. The Y loop counts in increments of 2, so the next number will be 10. When this number is output, Y/2+45 equals 50, which is the number 2 in ASCII terms. Notice that the Y loop counts from 8 to 24 in increments of 2. This is necessary to get the proper horizontal screen positioning. However, the same value is altered mathematically to cause the numbers 1 through 9 to be printed in steps of 1.

Let's talk about the monster face now. If one of your input answers is equal to the random number represented by numeric variable RN, this condition is picked up by line 400, and there is a branch to line 420. Here, the screen is cleared, and lines 450 through 550 draw the actual monster face. As you've already seen, this is a large block with eyes, nose, and mouth inserted. The block itself is drawn with HCHAR and VCHAR subprograms. Here, the ASCII block character is strung together with others of its kind. The eyes and nose are drawn with single ASCII block characters. The mouth is drawn with strings similar to those used to draw the head. Lines 430 and 440 produce a different color pattern for the monster face and also the loud growl. The latter is produced using the Call Sound subprogram and a −3 noise number.

When the monster is accessed, the program run is finished. It is necessary, however, to erase all previous information from our array in order to make way for a new program run. This is done in lines 560 through 580. This is a simple for-next loop which counts from 1 to 100 and assigns a value of 0 to each element position in array A. Lines 590 and 600 comprise a time delay loop and then the screen is cleared automatically. Lines 620 through 650 comprise the portion of the program that gives you your score. First, the value of I is altered to a percentage. I represents the number of doors you opened before the program was terminated by getting the monster. The scoring is given really as a percentage, although this is not specified on screen. For example, if you made nine choices, opening all nine doors before getting the monster, then I is equal to nine divided by nine (which is one) times one hundred. This means you got a perfect score, or 100. If the value of I is less than 15, this means you got the monster on the first try, so the value of I is reassigned in line 640 to 0. The score itself is printed in line 650. Line 660 then reassigns I to a value of 0 (in case this hasn't been done previously), and you are prompted to press the enter key to play again. When this is done, the program starts all over, and you or an opponent can try your luck again.

This is a very simple game to play, but a fairly complex one to write into a computer program. It is even more difficult to explain on a line-by-line basis. Just remember that the array holds the guesses previously made and feeds them into the portion of the for-next loop that prints null characters on the screen. Nine doors are always printed during each cycle, but the ones represented by your guesses are erased when the X loop has timed out.

Due to the graphics and sound effects, this program may be enjoyed by young and old alike. The monster face is quite large in proportion to the other on-screen data, so it seems to literally jump out at you, especially when accompanied by the low-frequency "growl". This is a simple game, but it is composed of the elements (graphics, sound effects, suspense) that make even highly complex games so enjoyable.

BINGO CALLER

This program is short and sweet, but believe it or not, it can completely take the place of those expensive bingo outfits that are often used by civic organizations to raise funds. You've probably seen the type that uses numbered ping pong balls, which float around on a cushion of air. These are often coupled with an electronic display that will light up the same number on a large electronic screen once the ping pong ball is placed in its correct slot. Some of these outfits cost thousands of dollars, but if you have the TI-99/4A computer and a large screen television receiver, you can accomplish the same thing for less money, and the computer version has no moving parts.

This program outputs any and all of the 75 alphanumeric possibilities associated with the game of bingo. You will recall that the numbers range from 1 to 75, with the first 15 placed under the B, the next 15 under the I, and so on. When this program is run, these 75 possibilities are chosen at random and are dis-

Listing 4. Bingo Caller

```
100  REM BINGO CALLER
110  REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES 3/6
/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  RANDOMIZE
140  CALL CLEAR
150  DIM A(75)
160  FOR X=1 TO 75
170  A(X)=X
180  NEXT X
190  Y=INT(RND*75)+1
200  IF A(Y)=0 THEN 190
210  IF A(Y)<16 THEN 220 ELSE
 240
220  Q$="B"
230  GOTO 340
240  IF A(Y)<31 THEN 250 ELSE
 270
250  Q$="I"
260  GOTO 340
270  IF A(Y)<46 THEN 280 ELSE
 300
280  Q$="N"
290  GOTO 340
300  IF A(Y)<61 THEN 310 ELSE
 330
310  Q$="G"
320  GOTO 340
330  Q$="O"
340  INPUT "PRESS (ENTER)":ER $
350  CALL CLEAR
360  PRINT "            ";Q$;A(
Y)
370  PRINT
380  PRINT
390  PRINT
400  PRINT
410  PRINT
420  PRINT
430  PRINT
440  PRINT
450  PRINT
460  PRINT
470  PRINT
480  A(Y)=0
490  GOTO 190
```

played one at a time at the center of the screen. When you press the enter key, another letter/number combination is displayed, and so on, until someone wins. Once a letter/number has been called, it will not be displayed again until a new game is started. This program allows the computer to act exactly like a commercial bingo calling machine and can allow many civic organizations who might not otherwise be able to afford one the opportunity to offer "low overhead" bingo.

While the program is short and fairly simple, it is a bit difficult to explain, so make certain you can view the program lines while reading this discussion. First, line 130 uses the randomize statement to assure a random combination of output call numbers. The number portion of the final alphanumeric output is contained in array A, which is dimensioned in line 150. This array will contain 75 different elements. Numbers from 1 to 75 are fed into the array by lines 160 through 180. This loop counts from 1 to 75 and inserts the value of X into the array with each cycle.

When the loop is timed out, it is necessary to read the numbers contained in the array at random. In other words, the numbers run from 1 to 75 sequentially in the array, but we will not be pulling them out of the array in this sequence. Line 190 assigns variable Y the value of RND*75+1. This is an integer value because the INT function was used, so Y will always be equal to a number between and including 1 and 75. It is not possible to predict what number this will be. Therefore, it's like pulling a number out of a hat or drawing a ping pong ball out of a container of ping pong balls. We'll skip line 200 for the present, but we'll come back to it in a bit.

Lines 210 through 330 read the value of A(Y). If Y is equal to 30, then A(Y) will be equal to 30, since the array received its number sequentially. Position A(1) will be equal to 1, and this number will be output when Y is equal to 1. One might wonder why an array is needed at all if the value of its element is the same as the value of the random number which accesses the date from this element. You'll see why the array is necessary shortly.

Lines 210 through 330 determine which letter (B, I, N, G, O) will go with the output from the array. Take line 210, for example. If the value of A(Y) is less than 16 (i.e., equal to 1-15), there is a branch to line 220, which assigns the letter "B" to the string variable Q$. Line 230 is then executed, which brings about a branch to line 340. Here, you are asked to press the enter key to print the bingo output. Line 350 clears the screen and line 360 prints a series of spaces followed by Q$ (in this case, "B"), and then followed by the value of A(Y). Lines 270 through 330 apply when the value of A(Y) is within other ranges of numbers. These lines assign the letters I, N, G, or O accordingly.

All right, why do we need the array? At this point, we really don't, but remember that I've only talked you through the output of a single alphanumeric bingo call. Without the array, we could go back to line 190 and ask the computer to output another random number that could be coupled with the appropriate letter. The random number will always be between 1 and 75, and the if-then statement lines would make all letter assignments. All of this is required for a proper bingo game.

However, suppose the random number generator assigns a value to Y which has been previously used. Assume this value is 10. The computer will display B10 at its output. If the same number crops up again, B10 will be called again. This doesn't work very well when playing bingo. In this example, the fact that a B10 comes up twice has little impact on the play, since the first time it occurred, this square on the card was supposedly covered. It is a nuisance, however, and it slows up the game because the computer is outputting what is now useless information, since this is a repeat call. The array is used to avoid such useless maneuvers. We need some method of removing a number from the 75 we have to choose from after it has been accessed once. Here's how it's done.

Let's assume that the random number assigned to Y is 10. Line 210 detects this situation and brings about branches which assign the letter B to Q$. Line 360 prints Q$ followed by A(Y) or "B 10". The print statements found in lines 370 through 470 simply cause the bingo call to be scrolled to the center of the screen. Line 480 is where the action actually takes place. In this line, the value of A(Y) is changed to 0. In other words, to get a 10 from the array, Y was equal to 10. Therefore, A(Y) is equal to 10. However, line 480 changes A(Y) or A(10) from a value of 10 to a value of 0. Line 490 then branches to line 190, where a new random number is output. Let's assume that this number is again 10. Here is where line 200 takes over. A(Y) or A(10) is no longer equal to 10. Its value has been reassigned to 0. Line 200 detects this and branches back to line 190. Here, another random number is output. In other words, line 200 says "Hey, this number has already been used, so I'm sending it back (to the random number generator) and asking for another one." All of the numbers originally entered into the array are equal to a value which is greater than 0. However, whenever any number is pulled from this array and displayed on the screen, it is then reassigned the value of 0. As more and more numbers are used, more and more of the array elements are equal to 0. Therefore, line 200 will be branching back to line 190 many more times near the end of a complete bingo call than at the beginning. Toward the end of the available elements in the array, one can begin to notice a very definite time lag in getting a printout. However, this has been almost completely rectified by placing the input statement in line 340 rather than between lines 480 and 490, as might be customary. As soon as the program is run, the bingo call number is determined. In other words, all of the numbers processing has been done. When you press the enter key, this number is printed; the value of A(Y) is reassigned to 0; and line 490 branches to line 190, where the next bingo call is determined. While you're reading the current call on the screen, the computer is already determining what the next call will be. I originally placed the input statement between what is now lines 480 and 490. Using this method, the first bingo call was displayed as soon as the program was run. When you pressed the enter key, line 490 branched back to line 190, and then you had to wait for the numbers to be processed for a new output. This was no problem during the first 20 or 30 calls, but lags began to crop up near the halfway point as more and more previously used numbers were output by the random number generator. The way the program is written now, the same lag occurs, but it is not noticeable in operation since the players are viewing the current call displayed on the screen or the human caller is announcing the number via a P.A. system. By the time all of this has taken place, a new output should be ready to be displayed as soon as the enter key is pressed. In other words, I have arranged the program so that the difficult processing portions take place at a time when the human beings are occupied with the previous call. The computer is working one call ahead of the human players rather than at their own speed.

To summarize, this program feeds 75 numbers into an array. It draws them out one at a time and after displaying them, reassigns their values to 0. Any output value of 0 causes a return to the random number routine, and a new number is output. This program can directly take the place of any other method of randomly determining bingo sequences and is quite easy to write. From a practical standpoint, there is a very real chance that a win will take place by the time half of the numbers contained in the array have been output. Occasionally, the calls will go to the three-quarter point, but rarely will any game go past this point. When bingo has been announced, the program is manually halted and then reran for the next game. If you want to play with this program, you can continue to press the enter key until all 75 numbers have been output. The last number will remain on the screen (along with its letter designator) and a continuous loop will be formed at lines 190 and 200. At this point, all of the numbers in the array are equal to 0, so line 200 will constantly branch back to line 190.

This program has been put to good use by several groups and clubs in my local community, allowing them to forego the usual mechanical or other electronic methods of bingo calling. This is an example of another random number program whose output cannot be predetermined.

CARD SHUFFLER/DEALER

The bingo game program discussed previously can also serve as the basis for a program that will shuffle and deal cards. In bingo, there are 75 number possibilities, while in most card games, there are only 52. In bingo, it was necessary to fit an appropriate letter with each number, while in a card game, the appropriate suits must be matched. When you break the two games down in this manner (mathematically), you can see the similarities, programming-wise, in the two types of games.

The simplest card shuffler program will consist of an array that will hold 52 elements sequentially numbered from 1 to 52. A random number generator routine is used to pull an element from the array at random. After this, the value of that element is reassigned to 0. As before, whenever an element that has a value of 0 is accessed, there is a branch back to the random number generator. In this manner, the same card is never dealt twice during a single game.

One might think that it's a bit simpler to write a program to shuffle and deal cards than it is to write a similar one called bingo. This assumption is based on the fact that bingo contains 75 numbers, whereas most card games require only 52 cards. While there are fewer cards to deal with than there are bingo calls, a practical card shuffler/dealer program will be far more difficult to write. In bingo, you have only two criteria to worry about. These are the value of the number and the matching letter. There, the letter B is matched to the first 15 numbers, the letter I to the next 15, and so on. The numbers which have been input to the array (1-75) may be used directly to represent bingo call numbers.

The situation is different in a card game, however. While there are 52 cards in the deck, they are not sequentially labeled from 1 to 52. There is a more complex combination with names in sets of four. Each card will have a number and a name, or a name and a name. For example, there is the 2 of hearts, which is identified by a number and a name. Then again, there is the ace of spades, which is identified by two names, ace and spades. We can use numeric variables, such as A(Y) to represent numerical values, but string variables must be used to print the names on the screen. This increases complexity quite a bit. Out of necessity, we must use a random number output of from 1 to 52. However, each number must be linked to the numeric and/or string naming of each card. In other words, the number 2 cannot necessarily represent the 2 of hearts. If it does, another number must represent the 2 of spades, another, the 2 of diamonds, and so on. Accessing a random number between 1 and 52 that cannot be repeated a second time is only part of what the program must accomplish. A large number of if-then statements are required to assign each number a particular playing card in a standard 52-card deck.

Listing 5. Card Shuffler/Dealer

```
100  REM CARD SHUFFLER/DEALE
R
110  REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES 3/6
/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  RANDOMIZE
140  CALL CLEAR
150  DIM A(52)
160  FOR X=1 TO 52
170  A(X)=X
180  NEXT X
190  Y=INT(RND*52)+1
200  IF A(Y)=0 THEN 190
210  IF A(Y)<10 THEN 220 ELSE
 250
220  Q$="SPADES"
230  A(Y)=A(Y)+1
240  GOTO 830
250  IF A(Y)<19 THEN 260 ELSE
 290
260  Q$="HEARTS"
270  A(Y)=A(Y)-8
280  GOTO 830
290  IF A(Y)<28 THEN 300 ELSE
 330
300  Q$="CLUBS"
310  A(Y)=A(Y)-17
320  GOTO 830
330  IF A(Y)<37 THEN 340 ELSE
 370
340  Q$="DIAMONDS"
350  A(Y)=A(Y)-26
360  GOTO 830
370  IF A(Y)=37 THEN 380 ELSE 400
```

Listing continued.

```
380  Q$="ACE OF SPADES"
390  GOTO 830
400  IF A(Y)=38 THEN 410 ELSE
 430
410  Q$="ACE OF CLUBS"
420  GOTO 830
430  IF A(Y)=39 THEN 440 ELSE
 460
440  A$="ACE OF DIAMONDS"
450  GOTO 830
460  IF A(T)=40 THEN 470 ELSE
 490
470  Q$="ACE OF HEARTS"
480  GOTO 8 0
490  IF A(Y)=41 THEN 500 ELSE
 520
500  Q$="KING OF SPADES"
510  GOTO 830
520  IF A(Y)=42 THEN 530 ELSE
 550
530  Q$="KING OF CLUBS
540  GOTO 830
550  IF A(Y) = 43 THEN 560 ELSE
 580
560  Q$="KING OF DIAMONDS"
570  GOTO 830
580  IF A(Y)=44 THEN 590 ELSE
 610
590  Q$="KING OF HEARTS"
600  GOTO 830
610  IF A(Y)=45 THEN 620 ELSE
 640
620  Q$="QUEEN OF SPADES"
630  GOTO 830
640  IF A(Y)=46 THEN 650 ELSE
 670
650  Q$="QUEEN OF CLUBS"
660  GOTO 830
670  IF A(Y)=47 THEN 680 ELSE
 700
680  Q$="QUEEN OF DIAMONDS"
690  GOTO 830
700  IF A(Y)=48 THEN 710 ELSE
 730
710  Q$="QUEEN OF HEARTS"
720  GOTO 830
730  IF A(Y)=49 THEN 740 ELSE
 760
740  Q$="JACK OF SPADES"
750  GOTO 830
760  IF A(Y)=50 THEN 770 ELSE
 790
770  Q$="JACK OF CLUBS"
780  GOTO 830
790  IF A(Y)=51 THEN 800 ELSE
 820
800  Q$="JACK OF DIAMONDS"
810  GOTO 830
820  Q$="JACK OF HEARTS"
830  INPUT "PRESS (ENTER)":ER
$
840  CALL CLEAR
850  IF SEG$(Q$,12,1)<>"" THE
N 880 ELSE 860
860  PRINT "       ";A(Y);Q$
870  GOTO 890
880  PRINT "       ";Q$
890  PRINT
900  PRINT
910  PRINT
920  PRINT
930  PRINT
940  PRINT
950  PRINT
960  PRINT
970  PRINT
980  PRINT
990  PRINT
1000  A(Y)=0
1010  GOTO 190
```

Let's take a look at the program itself. You will notice that the randomize statement is used in line 130 to assure an unpredictable output of numbers. The screen is cleared in line 140, and our single 52-element array is established in line 150. It bears the designation "A".

Lines 160 through 180 feed numbers into the array. This is handled in exactly the same manner as in the bingo program. X is stepped from 1 to 52, and each array element assumes a value of X. The value of array position 1 is 1, position 5 is 5, etc.

The random number is assigned to variable Y in line 190, and line 200 branches back to line 190 in the event that an array element returns a value of 0. This is exactly the method used at this point in the bingo program.

The actual number and name assignments are made in lines 210 through 820. Here's how it works. I simply let each number from 1 to 52 represent a different card. I wanted to take advantage of the actual numbers value output from the array as much as possible. In line 210, a branch occurs to line 220 if the output of the array A(Y) is less than 10. Here, Q$ is equal to the word SPADES. This means that any random number assigned to Y in the range of 1 to 9 will represent a card that is a spade. Line 230 reassigns the value of A(Y) to A(Y)+1. This means that if the random number output for Y is 1, this number will represent the 1+1 of spades, or 2 of spades. If the number output is 2, it will represent the 2+1 or 3 of spades. What this program does regarding the numbers portion of the card output is mathematically modify the number output by the random number generator.

Let's assume now that the output from the random number generator is 15. This is detected in line 250. Any number between 10 and 18 will represent a card of from 2 to 10 with a suit of hearts. Again, there is a branch to line 260, where Q$ is equal to hearts. The value of A(Y) is assigned to A(Y)−8, so for value of 15, A(Y)−8 equals 7. The card will be the 7 of hearts.

Lines 290 through 310 allow any number from 19 to 27 to represent a suit of clubs and the numeric values 2 to 10. Lines 330 through 350 form the same operations in the suit of diamonds for random numbers between 28 and 36. In each case, as soon as the suit name is established and the mathematical modification to A(Y) has been performed, there is a branch to line 830, which will eventually cause the card to be displayed on the screen.

This sequence is changed a bit when the output of the random number generator is equal to 37 or more. It's at this point that individual assignments of card names and suits must be made for the face cards, including aces. These are the cards in the deck which are not specified by a number. Rather, they are specified by a face name and a suit name.

Lines 370 and 380 allow a random number of 37 to represent the ace of spades. Lines 400 and 410 assign a 38 the value of the ace of clubs. This process continues through line 820, naming each face card and four suits for each.

At this point, all the processing work has been done. Line 830 contains an input statement that prompts the user to press the enter key. When this is done, the card that has been selected from our 52-card electronic deck is displayed on the screen. This is done in much the same manner as the bingo numbers were displayed. As before, as soon as one card is displayed, the processor is busy picking out another, but this latter card won't be displayed until you press the enter key again.

The means of displaying these cards by name and number (or by name and name for the face cards) involves a bit of complexity, as compared to the bingo game. It is necessary to display the modified output from the random number generator when a card of between 2 and 10 (on the face) is selected. However, when a true face card is keyed up, it is not desirable to display the random number value that caused it to be selected. In other words, if the number 43 is output from the random number generator, this will key up the king of diamonds. However, if we used the line found in program line 860 to display this card, the display would read:

```
43 KING OF DIAMONDS
```

To overcome this, line 850 is inserted. It uses the SEG$ function to test the value of Q$. What this line says is if the value of the 12th character in Q$ is *not* equal to a null character (in other words, no character at all), then branch to line 880 and simply print Q$. However, if the 12th character in Q$ is equal to a null character, then branch to line 860. You see, when you spell out ace of hearts, king of diamonds, etc., Q$ is longer in characters than for the numeric designations of Q$, which simply includes suit names alone. I checked all of the values for Q$ used to designate face cards and found that in every case, the 12th character in that string was a letter. With the lower value cards, (2 through 10), the string ended long before the 12th character was ever used. Therefore, there was no 12th character. In BASIC, we designate no character with two quotation marks back to back. When this program is executed, the SEG$ function scans the current value of Q$, and if there's no character there at all, it causes a branch to line 860, which prints the modified value of A(Y) followed by Q$. Again, when there is no character in the 12th position, this indicates a card with a numeric value of between 2 and 10. However, if SEG$ reads any character in the 12th character position, this indicates a face card, and A(Y) is not to be printed, since Q$ now contains the entire name of that card. Therefore, there is a branch to line 880, where Q$ alone is printed.

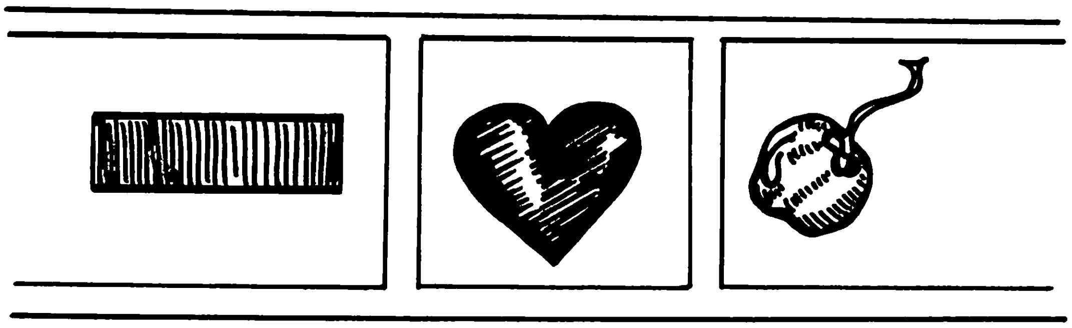
As was the case with the bingo program, print statements are used to cause the card to be displayed at the center of the screen. Line 1000 reassigns the value of A(Y) once again. This time, the value is 0, and this is fed back into the array. Assuming we were talking about the king of diamonds, which is accessed by the number 43 being returned by the random number generator, then forever after, array element position 43 will be equal to 0. If a 43 is output again by the random number generator, the 0 value is in effect. This is detected in line 200, and the program branches back to the random number generator for one more try.

Admittedly, this is not much of a game program in itself, although you can play a few simple games with it. The purpose of this program is to demonstrate the finer points of writing a program to shuffle cards and then be able to output the full 52-card deck by name to the screen. This same principle will be used in a later program that can be classified as a true computer card game. This program as shown can be quite useful, however, for those readers who might wish to write their own card game programs. This program can be input to your computer and stored on cassette or disk to be used as the basis for all other card games that involve a 52-card deck. With this *utility* program, hundreds of different card games can be programmed in a much shorter period of time, since the process of card shuffling, selection, and naming is fully taken care of.

ONE-ARMED BANDIT

Slot machine games can be programmed very easily on most microcomputers. Indeed, some may even have some of the symbols needed in their character set. This is not the case with the TI-99/4A, but it is quite easy to build up a number of graphic characters to represent the fruit on the real slot machines. This program uses a graphic heart, cherry, and bar to represent three slot machine windows. A win occurs when three of any one type of item line up in a row. Anything else is a loser. You may be surprised, however, at the number of times an in-line matchup occurs. This will certainly happen more often than on those machines in Vegas or Atlantic City.

Listing 6. One-Armed Bandit



```
100  REM ONE ARMED BANDIT
110  REM COPYRIGHT FREDERICK
     HOLTZ AND ASSOCIATES 2/2
8/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  RANDOMIZE
140  CALL COLOR(1,11,2)
150  CALL COLOR(2,10,2)
160  CALL COLOR(3,4,2)
170  CALL CLEAR
180  CALL SOUND(2400,-6,0)
190  X=INT(RND*3)+1
200  Y=INT(RND*3)+1
210  Z=INT(RND*3)+1
220  IF X=1 THEN 230 ELSE 250
230  A=34
240  GOTO 290
250  IF X=2 THEN 260 ELSE 280
260  A=40
270  GOTO 290
280  A=48
290  IF Y=1 THEN 300 ELSE 320
300  B=34
```

Listing continued.

```
310  GOTO 360
320  IF Y=2 THEN 330 ELSE 350
330  B=40
340  GOTO 360
350  B=48
360  IF Z=1 THEN 370 ELSE 390
370  C=34
380  GOTO 440
390  IF Z=2 THEN 400 ELSE 420
400  C=40
410  GOTO 440
420  C=48
430  REM HEART
440  CALL CHAR(34,"00001C3E7F
7F7F7F")
450  CALL CHAR(35,"0000387CFE
FEFEFE")
460  CALL CHAR(36,"3F1F0F0703
010000")
470  CALL CHAR(37,"FCF8F0E0C0
B00000")
480  CALL HCHAR(9,7,A)
490  CALL HCHAR(9,8,A+1)
500  CALL HCHAR(10,7,A+2)
510  CALL HCHAR(10,8,A+3)
520  REM CHERRY
530  CALL CHAR(40,"0000000000
1F3F7F")
540  CALL CHAR(41,"0000060810
204080")
550  CALL CHAR(42,"7F7F7F7F3F
3F1F00")
560  CALL CHAR(43,"E0F0F0F0F0
E0C000")
570  CALL HCHAR(9,16,B)
580  CALL HCHAR(9,17,B+1)
590  CALL HCHAR(10,16,B+2)
600  CALL HCHAR(10,17,B+3)
610  REM BAR
620  CALL CHAR(48,"0000000000
3F3F3F")
630  CALL CHAR(49,"0000000000
FCFCFC")
640  CALL CHAR(50,"3F3F3F0000
000000")
650  CALL CHAR(51,"FCFCFC0000
000000")
660  CALL HCHAR(9,24,C)
670  CALL HCHAR(9,25,C+1)
680  CALL HCHAR(10,24,C+2)
690  CALL HCHAR(10,25,C+3)
700  IF X=Y THEN 710 ELSE 730
710  IF X=Z THEN 720 ELSE 730
720  PRINT "YOU WIN THE JACKP
OT!!!"
730  INPUT "PRESS (ENTER) TO
PLAY AGAIN":A$
740  GOTO 170
```

Here's how the program works. First, the randomize statement is used in line 130 to make sure we get an unpredictable combination each time. Lines 130 through 160 assign different color patterns to character sets 1, 2, and 3, respectively. The Call Clear subprogram is used in line 170 to remove all previous information from the screen. Line 180 contains the Call Sound subprogram, which is supposed to simulate the sound of the windows turning. Actually, it doesn't sound very realistic, but a poor sound effect is better than none at all.

Lines 190 through 210 contain the RND functions and determine which graphic symbols will appear, and in what order, on the screen. Variables X, Y, and Z may be equal to any number from 1 to 3. Let's assume for the moment that X is equal to 1. This is determined in line 220, and so there is a branch to line 230, which assigns A the value of 34. At this point, there is another branch, this time to line 290. Here, the value of Y is tested. This is the second random number that is produced in line 200. Let's assume that Y is equal to 2. This condition will be detected in line 320, which will bring about a branch to line 330. Here, the variable B is assigned the value of 40. There is another branch to line 360, where the value of Z is determined and proper assignments made. Let's assume that the value of Z is 3. Line 360 branches to line 370 only when Z is equal to 1. Therefore, the else branch to line 390 is used. The else branch in line 390 is also used, since Z is not equal 2. The branch to line 420 assigns variable C the value of 48.

Variables A, B, and C are assigned certain numbers based upon the value of the random number assigned to X, Y, and Z. You will remember that X was given a value of 1 in this discussion, so why was A assigned the value of 34? The answer is because ASCII character 34 in line 440 begins the drawing of what will later be a graphic heart. The heart is actually drawn by four Call CHAR subprograms in lines 440 through 470. Notice that each of the lines assigns the hexadecimal information to a different ASCII character: 34, 35, 36, and 37. When A is equal to 34, you can see what will happen in lines 480 through 510 for the graphic images actually written on the screen. Lines 480 through 510 actually call the characters in lines 440 through 470 because of the value of A. In line 480, A will be equal to 34. Therefore, the graphic image contained in line 440 is printed on the screen. In line 490, ASCII character 35 is represented by A+1, or 35. This calls up the character produced in line 450. This continues until the entire graphic heart is drawn on the screen.

Now, we find that B is equal to 40 (in this example). This is the case because we allowed Y to be equal to 2. Look at lines 520 through 560. A graphic cherry is drawn by combining the Call CHAR subprograms in these lines. You will notice the character numbers specified a range from 40 to 43. In lines 570 through 600, the value of B is inserted into a Call HCHAR subprogram, and the four previous lines are printed on the screen. In this example, we allowed the value of C to be equal to 48, since Z was equal to 3, and the same basic routine is used in line 610 through 690 to print the bar on the screen.

Variables A, B, and C represent the first, second, and third graphic character positions placed side by side on the screen. If X was equal to 3, A would be reassigned a value of 48. In this case, line 480 would draw character A, or 48, which is found in line 620. It would draw characters 49, 50, and 51 as well, which would produce a complete bar at the left-hand character position.

Following the printout, lines 700 through 720 determine whether or not a win has occurred. Line 700 tests to see if the variable X is equal to the variable Y. If this is true, there is a branch to line 710, where a test is made to see if X is equal to Z. If it is, all three characters are identical and there is a branch to line 720, which announces that you have won the jackpot. On the other hand, if X is not equal to Y or X is not equal to Z, no match has occurred, and you are simply instructed to press the enter key again.

I have purposely limited this program to the display of only three graphic characters, although a true slot machine has many more possible combinations. You can expand upon this program, however, by allowing for more random numbers and by producing other graphic images in the same manner. This is a very entertaining game for young and old alike, and I think you will find the colorful display to be quite appealing.

GRAPHIC DICE

No computer game book would be complete without a program that simulated a dice roll. This one goes a step further, however, and actually draws the cubes on the screen so the display is far more realistic than those programs that simply print numbers on the screen.

It's quite simple to set up a dice routine. All that is involved is the outputting of two separate random numbers from 1 to 6. Indeed, many dice programs do this and nothing else. Some will add the two numbers together, and still others will build in program lines to react to certain sums, such as 2 (Snake Eyes), or 12 (Box Cars).

This dice program uses two random numbers that are integers and are within the range of 1 to 6, but it feeds this information to a long series of program lines that produce on-screen graphics that draw the dice faces to correspond to the numbers generated. Figure 4-5 shows a typical screen display.

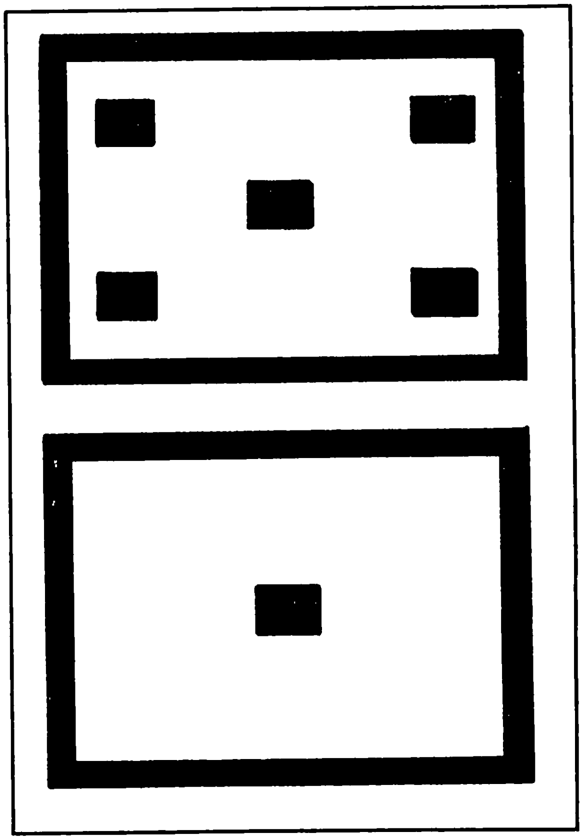
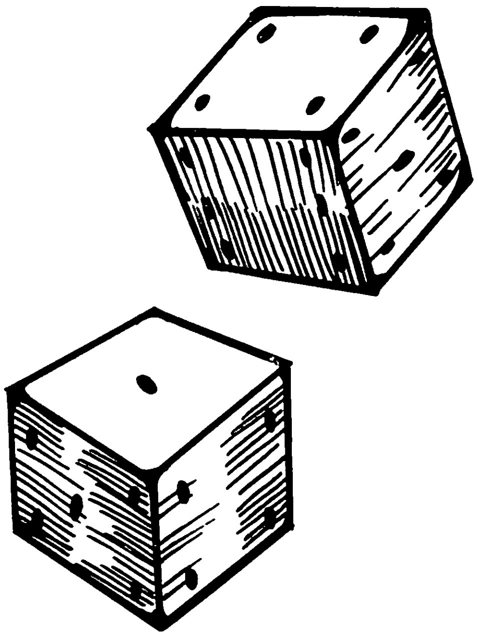


Fig. 4-5. Graphics dice screen display.

Here's how the program works. Lines 130 through 160 contain the familiar randomize, call clear, and RND statements and functions. The two random numbers are assigned to the variables S and T. Variable S represents the die that will appear at the left of the screen, while T is the right-hand die.

Lines 170 through 190 are used to draw the dice themselves and the dots which represent numbers. The character in line 170 draws a single dot, while the character in line 180 is the familiar filled-in block character which will

Listing 7. Graphic Dice



```
100  REM GRAPHIC DICE
110  REM COPYRIGHT FREDERICK
  HOLTZ AND ASSOCIATES 2/2
8/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  RANDOMIZE
140  CALL CLEAR
150  S=INT(RND*6)+1
160  T=INT(RND*6)+1
170  CALL CHAR(45,"183C7EFFFF
7E3C18")
180  CALL CHAR(34,"FFFFFFFFFF
FFFFFF")
190  CALL CHAR(46,"0000000000
000000")
200 CALL HCHAR(8,4,34,10)
210  CALL VCHAR(8,4,34,10)
220  CALL HCHAR(18,4,34,10)
230  CALL VCHAR(8,14,34,11)
240  REM SECOND DIE
250  CALL HCHAR(8,17,34,10)
260  CALL VCHAR(8,17,34,10)
270  CALL HCHAR(18,17,34,10)
280  CALL VCHAR(8,27,34,11)
290  FOR A=6 TO 12 STEP 3
300  CALL HCHAR(10,A,45)
310  CALL HCHAR(10,A+13,45)
320  CALL HCHAR(16,A,45)
330  CALL HCHAR(16,A+13,45)
340  CALL HCHAR(13,A,45)
350  CALL HCHAR(13,A+13,45)
360  NEXT A
370  BB=0
380  X=S
390  GOTO 420
400  X=T
410  BB=13
420  IF X=6 THEN 430 ELSE 480
430  GOTO 440
440  FOR AA=6 TO 12 STEP 3
450  CALL HCHAR(13,AA+BB,46)
460  NEXT AA
470  IF X=5 THEN 480 ELSE 550
480  CALL HCHAR(13,6+BB,46)
490  CALL HCHAR(13,12+BB,46)
500  CALL HCHAR(10,9+BB,46)
510  CALL HCHAR(16,9+BB,46)
520  IF X=4 THEN 530 ELSE 560
530  CALL HCHAR(13,9+BB,46)
540  GOTO 720
550  IF X=4 THEN 480
560  IF X=3 THEN 570 ELSE 640
570  FOR AA=6 TO 12 STEP 3
```

Listing continued.

```
580  CALL HCHAR(10,AA+BB,46)
590  CALL HCHAR(16,AA+BB,46)
600  NEXT AA
610  IF X=2 THEN 620 ELSE 650
620  CALL HCHAR(13,9+BB,46)
630  GOTO 720
640  IF X=2 THEN 570
650  IF X=1 THEN 660 ELSE 720
660  FOR AA=6 TO 12 STEP 3
670  CALL HCHAR(10,AA+BB,46)
680  CALL HCHAR(13,AA+BB,46)
690  CALL HCHAR(16,AA+BB,46)
700  CALL HCHAR(13,9+BB,45)
710  NEXT AA
720  IF BB=0 THEN 400
730  INPUT A$
740  GOTO 140
```

be strung together to draw the squares that represent each die. Line 190 contains a null character that will later be used to blank out unwanted information on the screen. This character simply overwrites in the color of screen background.

Lines 200 through 280 draw the two cubes on the screen. The first die is drawn in lines 200 through 230, and the second die is drawn in lines 240 through 280. Lines 290 through 360 put dots inside the cubes, which will later be selectively erased to produce a proper pattern. When the program is first run, Fig. 4-5 shows how the dice will appear momentarily. Notice that there are nine dots in the center of each cube. This pattern is necessary in order to generate all of the other possible patterns.

For the moment, let's skip to line 420 and assume that X is equal to 6. There is a branch to line 430, and then to line 440. This may seem a bit unusual, but I had originally included a special character line in program line 430. I decided to drop it for the presentation in this book. Therefore, line 430 simply moves the flow along.

Again, X is equal to 6, and we want to display a die on the screen with 6 dots showing. Lines 440 through 460 are a for-next loop, which counts from 6 to 12 in steps of 3. The on-screen position of the dots in the left-hand cube are 6, 9, and 12. This will be the output from the loop. The Call HCHAR subprogram in line 450 then prints a null character (erasure) at the position specified. We start out with 9 dots in the cube, but this for-next loop erases the center 3 at positions 13,6; 13,9; and 13,12.

Now let's go back to lines 370 through 410. While I could have written a program with separate lines to identify the two random numbers and then to erase dots in the appropriate square, it was far more efficient to use one set of program lines to do the job for both cubes. In line 370, during the first run, the variable BB is assigned the value of 0. Line 380 assigns the value of S (the left-hand die's random number) to the variable X. There is then a branch to line 420, where X is read. During this first run, we're talking about the left-hand die. You will notice that in line 450, the Call HCHAR subprogram uses the value of AA established by the loop, but also adds it to variable BB. Remember, at this stage, BB is equal to 0, so the screen position is AA+0, or AA. Now, skip all the way down to line 720, which is branched to in line 540 after the first cube is completed. It branches to line 400 when BB is equal to 0, which is presently the case. Line 400 now reassigns X to be equal to the value of T. Line 410 then reassigns the value of BB to 13. If T is also equal to 6, lines 440 through 460 will begin to erase dots, but this time at a position farther to the right that is equal to AA plus BB. Remember, the latter is now equal to 13. The second position specified will be 6 + 13; 9 + 13; and 12 + 13, or 19, 21, and 25. These are the horizontal positions of the dots in the second cube. When the 6 is formed again, there will be another branch to line 720, but BB is not equal to 0, so line 730 is executed. This produces a temporary halt in execution until the enter key is pressed. When this is done, there is a branch to line 140, which starts the dice roll all over again.

Various methods have been used to blot out the correct dots in order to be able to produce all numbers in standard fashion. In lines 470 through 530, a 5 is drawn in either cube, but this time, a for-next loop cannot be used and the null characters are placed at their positions using separate Call HCHAR subprograms. Again, you will notice the horizontal coordinate specification uses the BB variable to switch from the left cube to the right cube. This follows throughout the entire sequence, allowing each die to be represented in true-to-life fashion.

I purposely did not build additional test lines that might determine a win or loss based upon the rolling of a 7, 11, 2, or 12. This dice program is to be used for any and all purposes to directly take the place of mechanical dice. Certainly, you can use this program to play craps, but you can also use it to play Parcheesi, Monopoly, or for any other purpose that requires one or two dice. This is one of those programs which is as ideally suited for a child as for an adult. This program was written to be as efficient as possible. This means that unnecessary program lines are not included within loops. The completed dice images will appear rapidly, so you won't have to wait for 15 seconds or more in order to get a final output.

ROULETTE WHEEL

Roulette is one of the most ancient games of chance, although earlier forms of this game and the wheel with which it was played were quite different from those seen today. In any event, this program will simulate the roulette wheels of Las Vegas and Atlantic City and allow you to try your luck without the possibility of losing any money (or winning any either). This program automatically "stakes" you the sum of $1000, which you can bet any way you want as far as quantity is concerned. If you go broke, the computer has won; but if you're able to build your $1000 stake into $10,000 or more, you have "broken the bank" and beaten the machine.

In this version of roulette, you actually make four separate bets. Here, we are concerned with the actual number which lies in the range of 0 to 36; the color of the number, which can be black or red; whether the number is even or odd; and finally, whether it is high or low. In some versions of roulette, you may bet on any or all of these conditions. In this version, however, you must bet on them all. In other words, you are asked to choose the number on which the fictitious roulette marker will land, followed by the color (black or red). Any number which is divisible by 3 is red; all others are black. You are then asked to select whether the number will be even or odd.

Naturally, even numbers are evenly divisible by two; all others are odd. Finally, you bet on whether the number will be high or low. Low numbers constitute those from 0 to 18, while the high range from 19 to 36. Again, you must bet on all of these possibilities.

Here's how the payoff works. Each of the four bets constitute a possible win/loss situation. If you guess the correct number, the machine pays you 5 times the amount you bet. If you don't guess the correct number, you lose the amount of your bet. If you select the correct color, the machine pays you 3 times the amount of your bet. A loss here means you lose twice the amount of your bet. If you guess the odd/even sequence correctly, you win twice the amount of your bet, or if you are wrong, you lose the amount of your bet. Finally, getting the high/low sequence right nets you the amount of your bet, or if you don't guess right, loses you this same amount.

The most you can possibly win is 11 times your bet. This will occur only when you guess the correct number, color, odd/even, and high/low sequences. This is a billion to one shot. Most of the time, you will be correct on one or two and incorrect on others. Sometimes, you will win a little, while at other times, you will lose a little. Often, you'll end up neither winning nor losing any money. This occurs when you guess correctly at a few of the sequences and incorrectly at others. The combined wins and losses of a single spin may simply cancel each other out.

Again, you are allotted $1000 when the game begins. The computer will keep track of the amount you bet on each spin, along with your wins and losses. You will be apprised of your financial status at the end of each spin. The computer automatically adds and subtracts your wins and losses. If you run out of money, the computer displays a message which indicates your bankrupt status, and the game is ended. Likewise, if you build your "pot" into $10,000 or more, a screen message is printed indicating that you have beat the machine.

This program has many protective features built in. When you are asked to input a number between 0 and 36, you must do so. If you input a number outside of this range or no number at all, the program detects an erroneous input and will not move on until you have met the input requirements. This applies to the other three sequences within the betting regimen as well. Also, you cannot bet more money than you have. This program allows the machine to guard you as closely as the "wheel manager" would at a casino.

Listing 8. Roulette Wheel

```
100  REM ROULETTE WHEEL
110  REM COPYRIGHT FREDERICK
  HOLTZ AND ASSOCIATES 3/12/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  GOSUB 1350
140  CALL CLEAR
150  POT=1000
160  RANDOMIZE
170  X=INT(RND*37)
180  IF X/3=INT(X/3) THEN 190
ELSE 210
```

Listing continued.

```
190  Y$="RED"
200  GOTO 220
210  Y$="BLACK"
220  IF X/2=INT(X/2) THEN 230
ELSE 250
230  Z$="EVEN"
240  GOTO 260
250  Z$="ODD"
260  IF X<19 THEN 270 ELSE 29
0
270  I$="LOW"
280  GOTO 310
290  I$="HIGH"
300  CALL CLEAR
310  PRINT "NUMBER (0-36)?"
320  INPUT XX$
330  IF XX$<"0" THEN 300
340  IF XX$>"36" THEN 300
350  CALL CLEAR
360  PRINT "BLACK OR RED?"
370  INPUT YY$
380  IF YY$<>"BLACK" THEN 390
 ELSE 400
390  IF YY$<>"RED" THEN 350
400  CALL CLEAR
410  PRINT "ODD OR EVEN?"
420  INPUT ZZ$
430  IF ZZ$<>"ODD" THEN 440 E
LSE 450
440  IF ZZ$<>"EVEN" THEN 400
450  CALL CLEAR
460  PRINT "HIGH OR LOW?"
470  INPUT II$
480  IF II$<>"HIGH" THEN 490
ELSE 500
490  IF II$<>"LOW" THEN 450
500  CALL CLEAR
510  PRINT "YOU HAVE $";POT
520  PRINT
530  PRINT
540  PRINT
550  PRINT "HOW MUCH DO YOU W
ISH TO BET?"
560  INPUT BET
570  IF BET>POT THEN 580 ELSE
 630
580  CALL CLEAR
590  PRINT "YOU ONLY HAVE$";P
OT;"!!!"
600  FOR TD=1 TO 1000
610  NEXT TD
620  GOTO 500
630  CALL CLEAR
640  X$=STR$(X)
650  IF XX$=X$ THEN 660 ELSE
680
660  BET1=BET*5
670  GOTO 690
680  BET1=-1*(BET)
690  IF YY$=Y$ THEN 700 ELSE
720
700  BET2=BET*3
710  GOTO 730
720  BET2=-1*(BET*2)
730  IF ZZ$=Z$ THEN 740 ELSE
760
740  BET3=BET*2
750  GOTO 770
760  BET3=-1*(BET)
770  IF II$=I$ THEN 780 ELSE
800
780  BET4=BET
790  GOTO 810
800  BET4=-1*(BET)
810  BETT=BET1+BET2+BET3+BET4
820  CALL CLEAR
```

Listing continued.

```
830  PRINT "RESULT";
840  PRINT TAB(10);"SELECTION
"
850  PRINT
860  PRINT
870  PRINT X$;
880  PRINT TAB(10);XX$
890  PRINT Y$;
900 PRINT TAB(10);YY$
910  PRINT Z$;
920  PRINT TAB(10);ZZ$
930  PRINT I$;
940  PRINT TAB(10);II$
950  PRINT
960  PRINT
970  PRINT "YOU BET:$";BET
980  PRINT
990  PRINT
1000  PRINT
1010  IF BETT<0 THEN 1020 ELS
E 1040
1020  PRINT "YOU LOSE:$";-1*(
BETT)
1030  GOTO 1050
1040  PRINT "YOU WIN:$";BETT
1050  PRINT
1060  PRINT
1070  POT=POT+BETT
1080  IF POT<=0 THEN 1140
1090  IF POT>=10000 THEN 1240
1100  PRINT "YOU NOW HAVE $";
POT
1110  INPUT "PRESS (ENTER) TO
 CONTINUE.":ER$
1120  CALL CLEAR
1130  GOTO 170
1140  FOR TD=1 TO 1000
1150  NEXT TD
1160  CALL CLEAR
1170  PRINT "YOU ARE BANKRUPT
!!"
1180  PRINT
1190  PRINT
1200  PRINT "THE GAME IS ENDE
D!!!"
1210  PRINT
1220  PRINT
1230  END
1240  FOR TD=1 TO 1000
1250  NEXT TD
1260  CALL CLEAR
1270  PRINT "YOU HAVE BROKEN
THE BANK!!"
1280  PRINT
1290  PRINT
1300  PRINT
1310  PRINT "YOU HAVE $";POT
1320  PRINT
1330  PRINT
1340  END
1350  CALL CLEAR
1360  PRINT "WELCOME TO COMPU
TERIZED"
1370  PRINT
1380  PRINT "ROULETTE.  THIS G
AME ALLOWS"
1390  PRINT
1400  PRINT "YOU TO CHOOSE AN
Y NUMBER"
1410  PRINT
1420  PRINT "FROM ZERO TO 36,
 RED/BLACK,"
1430  PRINT
1440  PRINT "ODD/EVEN, AND HI
GH/LOW AS IN"
1450  PRINT
1460  PRINT "THE MECHANICAL G
AME.  THIS"
```

Listing continued.

```
1470  PRINT
1480  PRINT "VERSION GIVES YO
U 1000"
1490  PRINT
1500  PRINT "DOLLARS TO GAMBL
E WITH.  IF"
1510  PRINT
1520  PRINT "YOU WIN TEN THOU
SAND DOLLARS"
1530  PRINT
1540  PRINT "OR MORE, YOU BRE
AK THE BANK!"
1550  PRINT
1560  PRINT
1570  INPUT "PRESS (ENTER) FO
R PAY LIST.":ER$
1580  CALL CLEAR
1590  PRINT TAB(14);"WIN";
1600  PRINT TAB(23);"LOSE"
1610  PRINT
1620  PRINT "NUMBER";
1630  PRINT TAB(12);"5X BET";
1640  PRINT TAB(22);"BET"
1650  PRINT "BLACK/RED";
1660  PRINT TAB(12);"3X BET";
1670  PRINT TAB(22);"2X BET"
1680  PRINT "ODD/EVEN";
1690  PRINT TAB(12);"2X BET";
1700  PRINT TAB(22);"BET"
1710  PRINT "HIGH/LOW";
1720  PRINT TAB(12);"BET";
1730  PRINT TAB(22);"BET"
1740  PRINT
1750  PRINT
1760  PRINT "ANY NUMBER EVENL
Y DIVISIBLE"
1770  PRINT "BY THREE IS 'RED
', WHILE"
1780  PRINT "LOW NUMBERS RANG
E FROM 0-18."
1790  PRINT "HIGH NUMBERS ARE
 FROM 19-36."
1800  PRINT
1810  PRINT
1820  PRINT "PRESS (ENTER) TO
 PLAY."
1830  INPUT ER$
1840  RETURN
```

Looking at the program, you will immediately recognize the randomizing routine, which begins in line 160. Lines 160 and 170 assign X to a random number of between 0 and 36. This may not be apparent at first, since line 170 is handled a bit differently from other progams, especially those which produce computer dice rolls. You will notice here that the number multiplied by the RND function is 37 rather than 36. Previously, a line such as

```
170  X=INT(RND*6)+1
```

was used to simulate a dice roll. Here, X can be equal to any number from 1 to 6. However, in roulette, numbering is from 0 to 36. Therefore, it is necessary to be able to allow X to be equal to 0 as a minimum instead of 1. By multiplying RND times 37, we are assured that the number 37 will never crop up, since RND will always be a number that is less than 1. Using the integer function, a number such as 36.999 will be displayed as 36. On the other end, a number such as .999 will be displayed as a 0. Therefore, line 170 allows the variable X to always be equal to a number from 0 to 36.

Lines 180 through 210 test for the condition of the random number being assigned to a red or black color. Again, if the random number is evenly divisible by 3, it is assigned the red color; otherwise, it is black. Line 180 simply states that if X divided by 3 is equal to the integer of X divided by 3, branch to line 190. Here, the string variable Y$ is assigned the value of RED. Line 200 then branches to the next test sequence (even/odd). However, if X divided by 3 is not equal to the integer of X divided by 3, there is a branch to line 210, which assigns Y$ the value of BLACK. The same basic process takes place in lines 220 through 250 and in lines 260 through 290 to determine whether the random number is odd or even, high or low. You will notice that these assignments are made to different string variables. At the present time, the numeric variable X represents the number itself, while string variables Y$, Z$, and I$ represent red/black, even/odd, and low/high, respectively.

The result of each spin is actually determined before the betting takes place, although this information has not yet been displayed on the screen. When the program is initially run, there is a branch in line 130 to line 1350. The latter line begins the instructional sequence which prints all the game information on the screen. A later return statement at the end of the program branches back to line 140, where the screen is cleared. In line 150, the numeric variable POT is assigned the value of 1000. This is the amount of money you are staked when the game begins. Following this, the randomizing process described previously takes place.

The player is not asked for input until line 310 is encountered. Here, the player inputs his guess as to the number that will be output by the electronic roulette wheel. His guess is assigned to a string variable XX$. This is a bit unusual, since the number (X) output by the random number generator is a numeric variable. The string variable for the player input is used because it is easier to display on the screen in conjunction with the other bet sequence variables, which are, out of necessity, string variables. More on this later.

Lines 330 and 340 test for an improper input, such as the number being less than 0 or more than 36. If this occurs, there is a branch to line 300, where the screen is cleared and you are asked to input the number one more time. Lines 360 through 390, 410 through 440, and 460 through 490 allow you to input the other information needed for the bet sequence. Each input is assigned to a string variable: YY$ for black/red, ZZ$ for odd/even, and II$ high/low. Again, if you enter an improper choice, the program will not continue any further, but you will be given the opportunity to respond properly again and again.

Now that you have made your selection, it is time to indicate the amount you want to bet. You are prompted to do this in line 550, and line 560 asks for the input. In this case, the numeric variable BET is assigned to the amount you input. This can be any amount up to and including the amount you have in the pot. However, line 570 is there to make sure you don't bet more. Again, the variable BET is the amount you have bet, while the variable POT is the amount you presently have. If the former is larger than the latter, a message will appear on the screen telling you that you don't have that much and should try again.

It is now time to compare the player's guesses with the actual sequence of events that occurred during the spin. Before this can begin, it is necessary to do something about the X variable . This is a numeric variable that must be compared with a string variable (XX$). Again, it is easier in this program to display string variables in the final output sequence. When it is necessary to print both numeric and string variables on the same line, one has problems with uniform spacing between the two. A test line such as:

```
IF XX$=X
```

is not legal. We can extract the numeric value of XX$ using

```
XX=VAL(XX$)
```

However, this simply converts XX$ to a numeric variable, which is not what we want. What is needed is to convert the numeric variable (X) to a string variable. This is handled in line 640 using the STR$ function. This converts the numeric variable X into string variable X$. The value of X$ is the same as the value of X, so only the format has been changed, and it is now legal to compare the two, as is done in line 650.

Lines 650 through 800 compare the player's guesses with the actual output from the computerized spin. Variables BET1, BET2, BET3, and BET4 are used to represent the results of the sequence comparison. For example, line 650 tests for a condition of XX$ (player's guess) being equal to X$ (output from the random number generator). If the two are identical, the player hit the 1 in 36 odds and guessed the correct number. You will remember that this pays five times whatever value was bet. Line 660 then assigns the value of five times BET to BET1. However, if the two are not identical, line 680 takes over and assigns a value to BET1 that is equal to your bet times −1. In other words, if you bet $100 and missed the number, BET1 would be equal to minus $100. The remaining lines in this sequence test for other matches and make assignments to BET2, BET3, and BET4, respectively.

When all comparisons have been made, the various BET numbers are added together in line 810. Here, BETT (bet total) is equal to the sum of BET1, BET2, BET3, and BET4. Some of these will be equal to positive numbers, while others will be equal to negative numbers, depending on the outcome of the sequence. Line 820 clears the screen and the result sequence is then set up. A sample is shown in Fig. 4-6. On the left is the RESULT column, which indicates the actual outcome of the computer roulette spin. On the right, the SELECTION column indicates your guesses. The printing is handled using tab functions so that the information is neatly displayed on the screen as shown.

```
RESULT       SELECTION

36           24
RED          RED
EVEN         ODD
HIGH         LOW
```

Fig. 4-6. Result of one spin of the wheel in roulette.

Line 970 shows the amount you bet, while line 1010 checks to see if you have lost or won money. If you've lost, line 1020 prints the amount; whereas if you've won, the amount is printed by line 1040. Line 1070 adds the amount (positive or negative) of money involved in the last bet to the money you started with (POT). Lines 1080 and 1090 check for a win or a loss. The loss occurs when POT is equal to or less than 0. A win occurs when POT is equal to or more than $10,000. When either of these conditions occurs, branch to other program portions that indicate the win or loss status. If neither an overall win or loss has occurred, you are invited to press the enter key to play again. This is handled in line 1110, and when the enter key is pressed, there is a branch to line 170, where another spin of the wheel occurs.

This program is far more difficult to explain in standard English than it is to write in TI BASIC. It's not too much more difficult than writing a complete dice game program, and only a few extra comparisons need be made to determine the outcome. While there are 36 different numerical possibilities, these are only taken into account regarding the number guess. With all other possibilities, there are only two possible outcomes for each, which are easily determined (i.e., division by 3, division by 2, etc.).

A great many of the program lines are taken up with what I call safety routines to make certain that the player inputs correct information. The program can be greatly shortened without these safety lines, but it would not be as easy to play. For example, if you accidentally input LOP instead of LOW and there is no opportunity to change your guess, you can be sure that LOP will always result in an incorrect guess. However, with the safety lines built into this program, such an erroneous input is detected and the player is given the opportunity to try again.

This is a very enjoyable game and is easily modified to take into account the types of players who may be enjoying it. You can easily limit the amount of money that can be bet at any one time, and you can even set the program up so that the computer will allow you to go into debt to the "house" up to a certain amount of money.

All in all, it's a very interesting program and runs quite well on the TI-99/4A. The program is complete and needs nothing else to make it useful, although some players will undoubtedly want to add some sound effects and personal touches.

HANGMAN

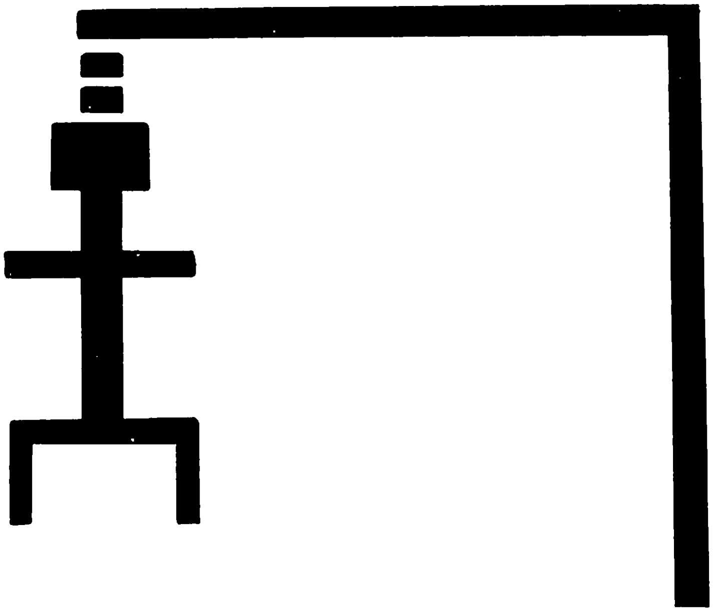
Hangman has been a popular game for centuries, and during the last half decade or so, it has proved very popular as a computerized game as well. This game, as presented here, is not significantly different from others that have been written for computers, at least as far as the on-screen display is concerned. However, the programming steps that were required to arrive at the finished game became quite involved. Using standard TI BASIC, it is a bit difficult to continually print information at one point on the screen and then update that information without erasing the screen entirely. Many dialects of BASIC offer locate statements or print @ statements, which will allow you to print a line of text at a certain position on the screen. Neither of these statements are found in TI BASIC, so it is necessary to find a different means of accomplishing the same task. It would be fairly simple to draw a gallows, a rope, and a figure at the end of the rope on the screen using TI BASIC. However, the game of Hangman requires that these graphics be produced on an element by element basis. After each element is formed, the player is given the opportunity to guess another letter. This involves on-screen text which will automatically cause the previous graphic image to be scrolled upward one line. This can be gotten around by clearing the screen after each guess and then redrawing the previously displayed graphics along with any new elements that might be added. For example, if you progress to a point in the game where you have a gallows, a rope, and a head on the screen, when you make your next guess and press the enter key, the screen is cleared. If your guess is wrong, the gallows, rope, and head will appear again along with the graphics that make up the body. If your guess is correct, the letter will be displayed on the screen.

In computer Hangman, it is necessary to display each correctly guessed letter on the screen at all times and at a position that will indicate its position within the word. For example, if the word is TRACK and someone guesses A, the letter A should be displayed in the third horizontal position on the screen to indicate where it falls within the word. This can be a real problem, even more of one than displaying graphic characters at certain positions on the screen. Again, this is due to the fact that there are no print @ or locate statements in TI BASIC. The closest alternatives are the subprograms Call VCHAR and Call HCHAR. However, these subprograms cannot be used directly with letters from the TI-99/4A character set. Rather, they must be used with the ASCII values that represent these characters. Therefore, it is necessary to convert the secret word input at the beginning of the game by the rival player to ASCII characters and convert the guessed letters to ASCII code as well. When this is done, the two are compared; and in the event of a correct guess, a Call HCHAR subprogram (Call VCHAR will work as well) is used to insert the ASCII character at a certain point on the screen. When displayed, the character is in alphabetical form. This was the hardest part of writing this program. The graphics part of it went along very well, but the rest of it consumed many hours of experimentation until a workable solution was finally arrived at.

Lines 100 through 490 are involved with printing a complete set of instructions on the screen. A large number of solo print statements are used to properly separate the text lines and display them most attractively. The set of instructions was so long that one screen was not adequate to hold all information. Therefore a time delay loop was inserted at the midway point. This is found in lines 320 and 330. Here, the numeric variable DELAY causes the computer to count from 1 to 500 before moving on to the next set of instructions beginning at line 340. This allows the player time to view the screen, read the information, and absorb the information before the scrolling begins as the loop times out and the additional instructions are printed.

The heart of the program begins in line 510, where all instructional information is cleared from the screen. A four-element array

Listing 9. Hangman



```
100  REM HANGMAN
110  REM COPYRIGHT FREDERICK
  HOLTZ  AND ASSOCIATES 3/2
0/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  CALL CLEAR
140  PRINT "WELCOME TO THE GA
ME OF COM-"
150  PRINT
160  PRINT "PUTER HANGMAN.  IT
 IS PLAYED"
170  PRINT
180  PRINT "IN A SIMILAR MANN
ER TO THE"
190  PRINT
200  PRINT"STANDARD VERSION.
 WHEN ASKED"
210  PRINT
220  PRINT "TO DO SO, ENTER AN
Y FIVE LET-"
230  PRINT
240  PRINT "TER WORD. NO REPE
TITION OF"
250  PRINT
260  PRINT "LETTERS IS PERMIT
TED."
270  PRINT
280  PRINT
290  PRINT
300  PRINT
310  PRINT
320  FOR DELAY=1 TO 500
330  NEXT DELAY
340  PRINT "WHEN YOU PRESS (E
NTER), THE"
```

Listing continued.

```
350  PRINT
360  PRINT "KEY WORD WILL DIS
APPEAR"
370  PRINT
380  PRINT "AND THE PLAYER WI
LL BE ASKED"
390  PRINT
400  PRINT "TO GUESS THE CORR
ECT LETTER."
410  PRINT
420  PRINT "A WIN OCCURS WHEN
 THE WORD"
430  PRINT
440  PRINT "IS GUESSED BEFORE THE
 VICTIM"
450  PRINT
460  PRINT "IS FULLY STRUNG U
P"
470  PRINT
480  PRINT
490  PRINT "PRESS (ENTER) TO
BEGIN"
500  INPUT Y$
510  CALL CLEAR
520  DIM C$(4)
530  CALL CHAR(34,"FFFFFFFFFF
FFFFFF")
540  CALL CHAR(35,"00000000FF
FFFFFF")
550  PRINT "TYPE ANY FIVE LET
TER WORD"
560  PRINT
570  PRINT "WHICH DOES NOT CO
NTAIN TWO"
580  PRINT
590  PRINT "OR MORE IDENTICAL
 LETTERS."
600  PRINT
610  INPUT A$
620  I=I+1
630  W$(I)=SEG$(A$,I,1)
640  IF I=5 THEN 660
650  GOTO 620
660  CALL CLEAR
670  INPUT "GUESS A LETTER":B
$
680  CALL COLOR(1,6,2)
690  FOR TY=1 TO R
700  IF B$=C$(TY) THEN 710 ELS
E 740
710  CALL CLEAR
720  PRINT "YOU ALREADY HAVE
THAT LETTER!"
730  GOTO 670
740  NEXT TY
750  CALL CLEAR
760  GOTO 1160
770  IF T>0 THEN 790 ELSE 670
780  T=T+1
790  REM HORIZONTAL GALLOWS
SECTION
800 CALL HCHAR(1,17,34,12)
810  IF T=1 THEN 670
820  REM VERTICAL GALLOWS SE
CTION
830  CALL VCHAR(1,29,34,20)
840  IF T=2 THEN 670
850  REM ROPE (THREE SECTION
S)
860  CALL VCHAR(2,17,35)
870  IF T=3 THEN 670
880  CALL VCHAR(3,17,35)
890  IF T=4 THEN 670
900  CALL VCHAR(4,17,35)
910  IF T=5 THEN 670
920  REM HEAD
```

Listing continued.

```
930  CALL HCHAR(6,16,34,3)
940  CALL HCHAR(7,16,34,3)
950  CALL HCHAR(8,16,34,3)
960  IF T=6 THEN 670
970  REM BODY
980  CALL VCHAR(9,17,34,8)
990  IF T=7 THEN 670
1000  REM LEFT ARM
1010  CALL HCHAR(11,15,34,2)
1020  IF T=8 THEN 670
1030  REM RIGHT ARM
1040  CALL HCHAR(11,18,34,2)
1050  IF T=9 THEN 670
1060  REM LEFT LEG
1070  CALL HCHAR(16,15,34,2)
1080  CALL VCHAR(17,15,34,2)
1090  IF T=10 THEN 670
1100  REM RIGHT LEG
1110  CALL HCHAR(16,17,34,2)
1120  CALL VCHAR(16,19,34,3)
1130  PRINT "YOU LOSE!!!"
1140  PRINT "THE CORRECT WORD
 WAS ";A$
1150  GOTO 1150
1160  FOR X=1 TO 5
1170  IF W$(X)=B$ THEN 1220
1180  NEXT X
1190  RQ=1
1200  GOSUB 1260
1210  GOTO 780
1220  S=ASC(W$(X))
1230  R=R+1
1240  E(R)=X
1250  G(R)=S
1260  FOR F=1 TO R
1270  CALL HCHAR(4,E(F)+2,G(F ))
1280  NEXT F
1290  IF RQ=1 THEN 1300 ELSE
1320
1300  RQ=0
1310  RETURN
1320  IF R=5 THEN 1350
1330  C$(R)=B$
1340  GOTO 770
1350  CALL CLEAR
1360  PRINT "YOU HAVE ESCAPED
 THE GALLOWS"
1370  PRINT
1380  PRINT
1390  PRINT "THE CORRECT WORD
 WAS ";A$
1400  PRINT
1410  PRINT
1420  PRINT "YOU WIN!!!!"
1430  PRINT
1440  PRINT
1450  PRINT
1460  END
```

designed to hold string information is created in line 520, while characters 34 and 35 are reassigned using Call CHAR subprograms in lines 530 and 540. Characters 34 and 35 from the character set are used to draw the on-screen graphics. Character 34 is the familiar "solid block", while character 35 is a half block.

Starting in line 550, you are instructed to type in any five-letter word that does not contain two or more of the same letter. In every case, the word must be five letters in length (neither more and nor less) and must not contain the same letter more than once. This word is the one that is to be guessed by another player. Therefore, this game is designed to be played by two individuals. When one is typing in the secret word, the other must look away. Alternately, you could use read/data statements to input a long list of five-letter words. Line 610 assigns the secret word to the string variable A$. Line 620 serves as a count line, which could just as easily be replaced with a for-next loop. Lines 620 through 650 are effectively a for-next loop, which counts from 1 to 5. Each time this loop cycles, W$(I) is assigned the value of SEG$(A$,I,1). Here's what is happening. The SEG$ function is used to read each of the five letters in A$. As the value of I increases, the W$ variable is relabeled, as in W$(1), W$(2), ... etc. The SEG$ function steps as well, as in SEG$(A$,1,1), SEG$(A$,2,1). . . etc. When variable I is equal to 5, there is a branch to line 660, where the screen is cleared. The purpose of this loop is to assign the five letters in the secret word (A$) to W$(1) through W$(5), respectively. This will be dealt with further in a bit.

After the screen is cleared, you are asked to guess a letter. This is where the second player comes in and begins to actually play the game, attempting to guess the secret word that was supplied by the first player. As soon as the letter, which is assigned to B$, is guessed and the enter key is pressed, line 680 sets up a colorful pattern combination on the screen for the most attractive display of the graphics. Forget lines 690 through 740 for right now. In line 750, the screen is cleared and line 760 branches to line 1160. This routine determines whether or not the input letter is the same as any of the letters in the secret word. You will remember that each letter was assigned to W$(1) through W$(5). Line 1160 begins a for-next loop, which counts from 1 to 5. In line 1170, the value of X is inserted in the parentheses following W$, and it compares each of the five stored letters with the input letter, B$. Let's assume that there is a match. This brings about a branch to line 1220.

Here's where the work begins. It was mentioned previously that TI BASIC offers no convenient way to print a letter in alphabetic form in a specific place on the screen. Line 1220 assigns to the variable S the ASCII value of the W$(X) string variable, which is the match with the letter guessed by the player. Line 1230 uses the variable R in a counting sequence, which will be described in a bit. In line 1240, the variable E(R) is assigned the value of X. Another variable G(R) is assigned the value of S.

Now, a for-next loop is formed that counts from 1 to the value of R. The first time around, the value of R will be 1, so the loop will make only one pass. Within the loop, there is a Call HCHAR subprogram which prints the ASCII character G(F) at a position on the screen of E(F) plus 2. I realize this is getting to be a bit difficult to understand, so to simplify things, remember that W$(X) is equal to the letter that has been correctly guessed. The value of X itself is equal to the position this letter occupies in the secret word. W$(X) is reassigned to an ASCII value, which is in turn reassigned to the variable G(F). This program uses X in several different ways to establish a letter position and indirectly, the letter itself. It is absolutely mandatory to convert the letter to its ASCII equivalent using the ASC function in line 1220.

When the loop in lines 1260 through 1280 times out, it tests for a value of RQ. RQ will be equal to 0 anytime a correct letter is guessed, but is assigned to a value of 1 in line 1190 when an incorrect guess occurs. If RQ is indeed equal to 1, there is a branch to line 1300, which reassigns its value to 0; but since we're talking about a sequence that involves the guess of a correct letter, the branch will actually be to line 1320, where a test is made for the value of R. If R is equal to 5, all of the letters in the word have been correctly guessed, and there is a branch to line 1360, which causes the screen to display a message indicating that you have won. If you have not guessed all the letters, however, line 1330 assigns the string value of B$ to the string array C$ and at position R. There is then a branch to line 770, which tests for a value of T. T is incremented in steps of 1 each time an incorrect letter is guessed. If your first guess was correct, there is a branch to line 670, which allows you to guess another letter.

Now let's see what happens when an incorrect letter is guessed. Again, the input letter is B$, and this value is tested in the loop contained in lines 700 through 740. You will remember that earlier a correct guess was committed to the C$ array. Line 700 compares the new letter with all letters contained in the C$ array. If there is a match, this means the letter has been previously guessed, and the prompt contained in line 720 is printed on the screen. The branch statement in line 730 gives you the chance to input another letter without losing your turn.

If there is no match, the loop times out and there is a branch in line 760, which takes us to line 1160. The same checks are made as have been previously discussed, but in this case, we are assuming an incorrect guess, so there will be no match found in line 1170. The loop will time out and RQ will be assigned a value of 1. There is then a GOSUB to line 1260. This starts a for-next loop that ends at line 1280. This loop prints the correctly guessed letters from previous turns on the screen. When the loop times out, line 1290 makes a test for the value of RQ. Since it is equal to 1, there is a branch to line 1300, which reassigns RQ the value of 0 again. The return statement in line 1310 branches to line 1210. There is then another branch to line 780.

Line 780 begins the graphic portion of the program and is activated whenever an incorrect guess occurs. In line 780, the variable T is incremented in steps of 1 each time an incorrect guess occurs. REM statements have been included in lines 790 through 1100 to show which program lines draw the various body parts. For instance, line 790 indicates that the following lines are used to graphically depict the horizontal portion of the gallows. This is drawn when the first incorrect guess occurs. Line 810 tests for the number of wrong guesses. If T is equal to 1, as it is in this case, this means that only one incorrect guess has occurred. Therefore, only the horizontal gallows section need be drawn. There is then a branch to line 670, which allows for another guess.

For the sake of clarifying the discussion, let's assume that the next guess is incorrect. As before, all the tests and branches will take place until we end up again at line 780. The screen has been cleared, so the horizontal gallows section drawn previously is erased. Now, T is incremented by 1, so it is equal to 2. Line 800 draws the horizontal gallows section, and line 810 then tests for a condition of T being equal to 1. This is no longer the case, as T is equal to 2. Therefore, line 830 is executed, which draws the vertical gallows section. Another test line for T is found in line 840. Here, when T is equal to 2, the graphic drawing process stops, and there is a branch to line 670 which allows you to guess another number. Remember, each time a new guess is entered, the screen is cleared and is then almost instantly rewritten with any new additions necessary. For instance, if the word is track and you have guessed two letters correctly (T and R), and two incorrectly, then at the start of your fifth guess, the letters TR will appear at the lefthand side of the screen and the vertical and horizontal gallows sections will be seen at the right.

Let's assume that your next guess is incorrect. The screen is completely erased. The letters "TR" are then printed again, followed by the printing of the horizontal gallows section, the vertical gallows section, and due to the wrong guess, a section of rope. The latter has been added to the previous screen, but it was necessary to erase and rewrite the entire screen due to the scrolling effect. In other words, if anything on the screen is to be changed in regard to the display of the letters correctly guessed and the graphic gallows and figure, the entire screen must be erased and rewritten with the new additions.

You are allowed 11 incorrect guesses before you actually lose the game. The last graphic segment to be drawn will be the right leg, which is produced in lines 1110 and 1120. When this occurs, lines 1130 and 1140 are executed, indicating that you have been "strung up". An endless loop is created in line 1150 to allow the image on the screen to hold its graphic appearance. To play another game, you must manually halt execution (FCTN & 4) and then run the program again.

Hangman is never an easy program to write, regardless of the machine being used for execution. By the same token, it's not an extremely difficult program in most instances. However, using standard TI BASIC (as opposed to Extended BASIC for instance), Hangman is a great challenge for the TI-99/4A programmer. The same applies to any home computer that displays information by scrolling from bottom to top rather than by printing from top to bottom, as is standard with most business and personal computers. Therefore, it is harder to accurately set text and graphic information using the BASIC language found in home computers. However, with a bit of resourcefullness and imagination and a large heaping of knowledge about TI BASIC, these difficulties can be overcome, although programming time may be doubled. In the end, however, this Hangman program runs in much the same manner as far as the player is concerned as those written for more powerful machines using a dialect of BASIC that allows for easy placement of characters on the screen.

TIC-TAC-TOE (Manual)

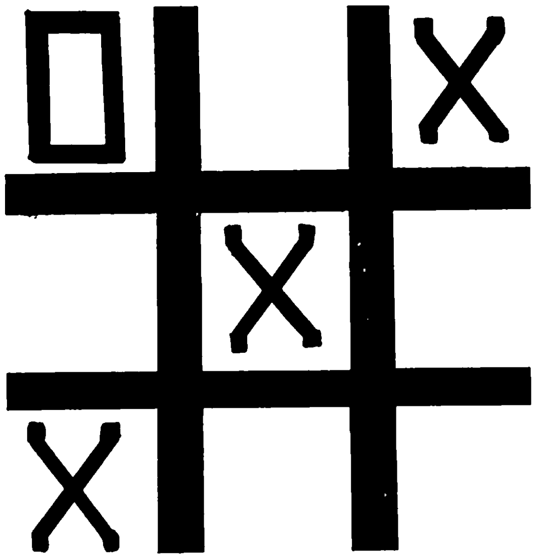
The game of tic-tac-toe has been enjoyed for hundreds upon hundreds of years, and in recent times, it has been committed to the microcomputer using hundreds and hundreds of different programs. The version of tic-tac-toe shown here can certainly be classified as low level, since you don't actually play the computer, but rather, another human player. This program allows the TI-99/4A computer and its monitor screen to serve in place of paper and pencil, which would normally be used to implement the mechanical version of this ancient game.

When the program is run, the monitor screen will display the dual crosshatch that represents the game board. Each of the nine squares will be identified by a number of from 1 to 9. In order to fill in a square, all you need do is hit the numbers key that corresponds to the block you wish to occupy. In this game, X always has the first move.

While relatively simple in nature, this program has several built-in protection features that greatly eliminate the chance of input errors through accidental pressing of erroneous keys. Once a square has been filled in, it cannot be filled in again by another player. The Xs and Os which make up the moves of the two players are always handled on an alternating basis. In other words, the first player to input a number will cause an X to appear in the appropriate box on the screen. The next number that is entered will cause an O to be displayed, followed by another X for the next number. As is the case with most tic-tac-toe games run on a computer, a win does not cause a solid line to be drawn through the series of boxes which constitute that win. This can be done, but at the expense of many more program lines. As a matter of fact, this highly simple program does not even register a win in its present form. This could also be accomplished by adding a win detect subroutine. However, the idea of this game is to allow two players to enjoy the game of tic-tac-toe without the use of paper and pencil.

Realistically speaking, tic-tac-toe is a game for children, since two adults will almost always play to a tie. There just aren't as many variables as there are in chess or checkers, so the game quickly becomes boring. However, children can delight in this game for hours on end and gain experience with a microcomputer as well.

Listing 10. Tic-Tac-Toe



```
100  REM TIC TAC TOE
110  REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES #/2
3/83
120  REM PROGRAM RUNS IN TI-
BASIC
130  CALL CLEAR
140  CALL CHAR(33,"FFFFFFFFFF
FFFFFF")
150  CALL VCHAR(3,13,33,20)
160  CALL VCHAR(3,20,33,20)
170  CALL HCHAR(9,7,33,20)
180  CALL HCHAR(16,7,33,20)
190  CALL HCHAR(6,9,49)
200  CALL HCHAR(6,16,50)
210  CALL HCHAR(6,24,51)
220  CALL HCHAR(13,9,52)
```

Listing continued.

```
230  CALL HCHAR(13,16,53)
240  CALL HCHAR(13,24,54)
250  CALL HCHAR(20,9,55)
260  CALL HCHAR(20,16,56)
270  CALL HCHAR(20,24,57)
280  C=C+1
290  IF C/2=INT(C/2) THEN 300
ELSE 320
300  X=79
310  GOTO 330
320  X=88
330  CALL KEY(0,I,S)
340  IF S=0 THEN 330
350  IF I=49 THEN 510
360  IF I=50 THEN 560
370  IF I=51 THEN 610
380  IF I=52 THEN 660
390  IF I=53 THEN 710
400  IF I=54 THEN 760
410  IF I=55 THEN 810
420  IF I=56 THEN 860
430  IF I=57 THEN 910
440  GOTO 330
450  FOR T=1 TO 50
460  NEXT T
470  GOTO 280
480  REM  1
490  D=D+1
500  IF D>1 THEN 330
510  REM  1
520  B=B+1
530  IF B>1 THEN 330
540  CALL HCHAR(6,9,X)
550  GOTO 450
560  REM  2
570  E=E+1
580  IF E>1 THEN 330
590  CALL HCHAR(6,16,X)
600  GOTO 450
610  REM  3
620  F=F+1
630  IF F>1 THEN 330
640  CALL HCHAR(6,24,X)
650  GOTO 450
660  REM  4
670  G=G+1
680  IF G>1 THEN 330
690  CALL HCHAR(13,9,X)
700  GOTO 450
710  REM  5
720  H=H+1
730  IF H>1 THEN 330
740  CALL HCHAR(13,16,X)
750  GOTO 450
760  REM  6
770  J=J+1
780  IF J>1 THEN 330
790  CALL HCHAR(13,24,X)
800  GOTO 450
810  REM  7
820  K=K+1
830  IF K>1 THEN 330
840  CALL HCHAR(20,9,X)
850  GOTO 450
860  REM  8
870  L=L+1
880  IF L>1 THEN 330
890  CALL HCHAR(20,16,X)
900  GOTO 450
910  REM  9
920  M=M+1
930  IF M>1 THEN 330
940  CALL HCHAR(20,24,X)
950  GOTO 450
```

From an adult standpoint, the most interesting thing about this program is not in the way it runs so much as the way it is constructed on a line-by-line basis. When the program is initially activated, the screen is cleared, and a block character is established in line 140. It is assigned to ASCII code number 33. The two vertical lines that constitute half of the tic-tac-toe gaming board are created by program lines 150 and 160. Each of these strings is formed by putting 20 block graphic characters together, starting at positions 3,13 and 3,20, respectively. The horizontal lines which complete the crosshatch are produced by lines 170 and 180 using the same method. In this latter case, Call HCHAR subprograms are used to string the 20 block series horizontally on the screen.

Lines 190 through 270 print the numbers in the nine blocks. The screen positions in each of these subprograms place the number at the center of the block. The numbers 1 through 9 are printed on the screen by using their ASCII codes in each of the Call HCHAR subprograms in lines 190 through 270. For example, in line 190, ASCII code 49 represents the number 1. ASCII code 50 represents the number 2, and so on.

When line 270 has been executed, the computerized game board is completely initialized. The next portion of the program deals with player input, and more specifically, places the Xs and Os as required points on the screen.

Lines 280 through 320 determine whose move it is. The X will always be displayed first and the O second. Therefore, X will always move on odd numbers while O will be displayed on even numbers. Line 280 assigns the variable C a value of itself plus 1. This is a counting routine, so C will increase in steps of 1 each time line 280 is executed. When the program is first run, the screen is initialized, and C is equal to 0 plus 1. Line 290 tests for an even or odd condition. It divides the value of C by 2 and if this is equal to the integer value of C divided by 2 (in other words, C/2 does not leave a fraction), there is a branch to line 300. Here, the variable X is assigned a value of 79, which is the ASCII code for the letter O. However, when C is equal to 1 or any odd number, C divided by 2 will not be equal to the integer value of C divided by 2, so the branch is to line 320. Here, X is assigned the value of 88, which is the ASCII code for the letter X. This is the value X will have when the first player inputs his move.

To prevent screen scrolling each time an input is asked for, I used the Call Key subprogram, which is similar to the INKEY$ statement in other dialects of BASIC. The Call Key subprogram in TI BASIC monitors the keyboard and assigns a value of 0 to S when no keys are pressed. When any key on the keyboard is pressed, the value changes to positive 1 or negative 1, depending on which key is utilized. At the same time, variable I in line 330 is assigned the ASCII value of the key that was pressed. The 0 designation preceding I and S simply indicates that the entire keyboard is to be monitored. Other numbers may be input here to monitor either the left or right side of the keyboard.

A continuous loop is established in line 340 as long as no key is pressed. Given this situation, the variable S in line 330 will be equal to 0, so there will be a branch to line 330. Lines 330 and 340 will continue to be executed in the loop until some key is pressed.

When this occurs, the ASCII value of the key will be assigned to the variable I contained in line 330. This breaks the loop and test lines 350 through 430 are executed. These test for the value of I, which should be in a range of from 49 to 57. These numbers are ASCII codes that represent the printed numbers 1 through 9 in the blocks on the screen. If for some reason a player hits a key other than those representing the numbers 1 through 9, I will not be equal to any of the numbers specified in lines 350 through 430. The GOTO statement in line 440 will then be executed. This branches back to line 330 again, and the endless loop between lines 330 and 340 is once again established. The player who provided the erroneous input does not lose his turn, however, and is allowed to select a correct number key.

For the sake of this discussion, let's assume that the screen has been initialized and the first player inputs the number 9, meaning that he wishes to place an X in block number 9 on the tic-tac-toe game board. Upon pressing the key, the endless loop in lines 330 and 340 is broken and test lines 350 through 430 go to work to locate the number and branch appropriately. The key representing the number 9 returns the ASCII code of 57, and this is detected in line 430, which branches to line 910. Lines 510 through 940 contain Call HCHAR subprograms whose coordinates match the coordinates of the screen positions where the nine numbers have been printed. The variable X contained in each of these subprograms is assigned an ASCII value of 88 in this particular case. During the next move, X will be equal to 79, which represents the letter O.

The branch to line 910 accesses the subroutine which will print the character represented by the ASCII number assigned to the variable X at the position on the screen which is currently occupied by the number 9. First, however, a counting sequence is encountered at line 920, which assigns variable M the value of 1. Line 930 tests the value of M to make sure it is not equal to more than 1. Finally, line 940 prints the letter X over top of the number 9 in the tic-tac-toe box. Remember, the variable X in line 940 is not the same as the letter X, which is printed on the screen. In this particular case, the variable X represents the letter X, but during the next move, the variable X will represent the letter O.

As soon as the character has been printed in the appropriate box, line 950 branches to line 450. This line and the next form a short time delay loop. This prevents players from accidentally activating an erroneous move by pressing more than one key simultaneously. When the loop times out, there is a branch to line 280, and the variable C is stepped by another increment of 1. The variable C is now equal to 2, and this is detected in line 290, which branches to line 300. The variable X is now assigned the ASCII code value of 79, and again, the Call Key subprogram loop is entered. It is the next player's move, and he may select any number from 1 to 9 that has not already been chosen and print his character in the appropriate box.

But what if the second player also chooses the number 9? You will remember that this box has been filled by the first player. Here's what happens. When the number 9 is pressed by the second player, this is detected by line 430, and again, there is a branch to line 910. However, line 920 steps the variable M once again. M is now equal to 2, since it was equal to 1 on the last go-round. Line 930 reads the fact that M is equal to a value which is greater than 1, so there is an immediate branch to line 330. The Call Key subprogram loop is entered again, but the player has not lost his turn. This same protective routine applies to the nine other subroutines which are used to print characters over top of the block numbers. Once a block has been selected, it cannot be selected again during the same game. This applies to the player who originally filled the box, as well as to the other player.

This program uses no prompts as such. You simply run it and begin making moves. As soon as one player has moved, the other may do likewise immediately. Again, this program is not designed to detect a tic-tac-toe win, so it will be necessary for the players to determine this.

The use of the Call Key subprogram helps avoid a great deal of screen reprinting that is necessary when using input statements. The previous Hangman program used input statements, and it was necessary to rewrite the entire screen after every move. Tic-tac-toe could have been handled in the same manner, but the Call Key subprogram has allowed for a shorter programming time to arrive at the desired on-screen effect. The Hangman program could be written in the same manner, although it would be necessary to make considerable modifications due to the increased complexities involved.

LETTER CONFUSION

Letter Confusion is an extremely interesting program, from both the player's and the programmer's point of view. The game is potentially unlimited and may be reseeded with new words at any time. The idea of the game is to unscramble a combination of letters that appear on the screen in order to come up with the correct word. In some cases, it is possible to make two or more words from the letters (rare), but there is always only one correct answer. A scorekeeping routine is included in the program, so once all of the scrambled words have been presented and the player has input his best guess for each, a final tally appears, indicating the number of correct answers as well as the number that were incorrect.

This program is designed to be enjoyed by persons of all ages and educational levels. No modifications are required to switch from a juvenile player to an adult. Three difficulty levels are built into this program. The first one is extremely elementary and is designed for children. The second one is probably the one that most adults will start with and is rated intermediate in difficulty. The third level is for the experts and includes some extremely complex scrambles.

Looking at the program, you can see that a string array composed of ten elements is established in line 130. Ten elements were chosen here, since the maximum length of any word is ten letters. However, if you wish to include longer words, you can simply increase the dimension figure. The rest of the program can remain the same. However, scrambled words of more than ten letters will be nearly impossible to decipher.

Line 140 brings about a branch to the subroutine found in lines 870 through 1120.

Listing 11. Letter Confusion

```
100  REM LETTER CONFUSION
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES 4/5/
83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  DIM A$(10)
140  GOSUB 870
150  CALL CLEAR
160  PRINT "WHAT DIFFICULTY L
EVEL"
170  PRINT "(1,2 OR 3)?"
180  INPUT D
190  FOR G=1 TO D
200  READ B$
210  NEXT G
220  L=LEN(B$)
230  GOTO 290
240  FOR GG=1 TO 3
250  READ B$
260  IF B$="END" THEN 760
270  NEXT GG
280  L=LEN(B$)
290  FOR X=1 TO L
300  A$(X)=SEG$(B$,X,1)
310  NEXT X
320  CALL CLEAR
330  RANDOMIZE
340  RN=INT(RND*L)+1
350  IF A$(RN)="0" THEN 340
360  PRINT A$(RN):
370  A$(RN)="0"
380  T=T+1
390  IF T=L THEN 410
400  GOTO 340
410  PRINT
420  PRINT
430  PRINT
440  PRINT
450  PRINT "GUESS THE WORD."
460  PRINT
470  PRINT
480  INPUT C$
490  CALL CLEAR
500  IF C$=B$ THEN 510 ELSE 5
40
510  PRINT B$;" IS CORRECT!!"
520  SC=SC+1
530  GOTO 570
540  PRINT "WRONG!!!"
550  SW=SW+1
560  PRINT "THE CORRECT ANSWE
R IS ";B$
570  PRINT
580  PRINT
590  PRINT
600  PRINT
610  PRINT "PRESS (ENTER) TO
CONTINUE"
620  INPUT ER$
630  T=0
640  GOTO 240
650  DATA CAT,HORSE,ALLIGATOR
,DOG,MOUSE,TERRIBLE
660  DATA FROG,TADPOLE,SURGEO
N,DRIP,FABLE,GOBLET
670  DATA SAND,BASTE,BALANCED
,HAT,CROSS,MARSHY,DRAW,GNAT,
ELEPHANT
680  DATA JOG,BRAG,QUICKLY,BA
SH,TRAPPED,INFANTILE,GEE,HAR
DLY,HONESTLY
690  DATA COST,REASON,CONQUER
ED,VAT,BRASH,VEXATED,WAG,TAL
```

Listing continued.

```
ON,GHOSTLY
700  DATA BUS,BANISH,SANCTIFY
,JOB,CARTS,COURTESY,CLUB,QUE
EN,CANNISTER
710  DATA ROB,CAUGHT,NAMELESS
,CAR,REBEL,HARRANGUE,LOB,HEDG
E,STUDENTS
720  DATA HUB,ZERO,XENOPHOBE,
WAY,TRASH,HAPPENING,KIT,F
RESHET,JOBLESSNESS
730  DATA WART,MARKER,LIQUOR,
FAT,LASHED,VILLIFY,MASS,CARB
ON,AIRPLANE
740  DATA SAT,RAGE,NARCOTIC,HA
ND,BEATS,CRIMINAL,POP,UNDER,
COMPUTERS
750  DATA CUT,MAKES,HACKLES,R
ICE,TREMBLE,WARTHOG,KILL,ZES
TY,JINGLES,END,END,END
760  CALL CLEAR
770  PRINT "THE GAME IS OVER"
780  PRINT
790  PRINT
800  PRINT "YOUR SCORE IS";SC
810  PRINT "CORRECT AND";SW;"
WRONG"
820  PRINT
830  PRINT
840  PRINT
850  PRINT
860  END
870  CALL CLEAR
880  PRINT "WELCOME TO THE GA
ME OF"
890  PRINT
900  PRINT "WORD SCRAMBLE. TH
E COMPUTER"
910  PRINT
920  PRINT "WILL PRESENT SCRA
MBLED WORDS"
930  PRINT
940  PRINT "FOR YOU TO DECIPH
ER. YOUR"
950  PRINT
960  PRINT "SCORE WILL BE GIV
EN AT THE"
970  PRINT
980  PRINT "END OF THE GAME."
990  PRINT
1000  PRINT
1010  PRINT "THERE ARE THREE
DIFFICULTY"
1020  PRINT
1030  PRINT "FACTORS. INPUT A
 ONE (1) FOR"
1040  PRINT
1050  PRINT "EASY, TWO (2) FOR
MODERATE,"
1060  PRINT
1070  PRINT "OR THREE (3) FOR
HARD. GOOD"
1080  PRINT
1090  PRINT "LUCK!!!"
1100  FOR LAG=1 TO 2500
1110  NEXT LAG
1120  RETURN
```

This subroutine prints a set of instructions on the screen. These instructions are displayed for 30 seconds or so due to the time delay for-next loop established in lines 1100 and 1110. When the loop times out, the return statement in line 1120 is executed, and there is a branch to line 150, where all information is cleared from the screen. The player is then asked to input the difficulty factor, which is specified by the numbers 1, 2, and 3. 1 represents the easiest level, while 3 is expert. The difficulty number is assigned to variable D.

At this point, it is necessary to discuss lines 650 through 750. These are the data statement lines and contain the words that are to be scrambled. The actual scrambling routine will be discussed a bit later. For now, it is only necessary to know that these data statements contain the words in groups of three. For example, in line 650, the word CAT will be displayed on the screen in scrambled form if 1 is chosen as the difficulty factor. This is the simplest of the three words. If an intermediate level is chosen (2), the word CAT will be ignored, while the word HORSE will be scrambled. If an expert level is chosen (3), the word ALLIGATOR will be scrambled. This holds true throughout the data statement lines. Once a word has been chosen, the next two words will be skipped on the following turn. Words are selected in increments of three.

With this arrangement understood, our discussion picks up again at program line 190. You will recall that the variable D represents the difficulty factor, and line 190 begins a for-next loop which counts from 1 to the value of D. Let's assume that you chose the lowest difficulty factor, or 1. This means that the loop will cycle only once and then time out. Line 200 (within the loop) contains the read statement, which pulls the first word from the data statement. You will remember that this is the word CAT and represents one word of a three-word set (the one that is the least difficult).

The loop times out and variable L is then assigned a value which is equal to the number of characters in B$. This is accomplished using the LEN function. The following lines scramble the letters in the word.

Before going further, let's assume that you chose a difficulty factor of 2 instead of 1. Returning to line 190, the loop will now count from 1 to D, the latter having a value of 2. Therefore, the loop makes two passes instead of one. During the first pass, the read statement accesses the first word in the data statement (CAT), but the loop does not time out since it must make one more pass. During the second pass, the read statement accesses the word HORSE, which is the first word to be scrambled in level 2 difficulty. At this time, the loop terminates, and line 220 assigns the length of B$ to the variable L. Should an expert level be chosen (3), the loop will cycle three times. This is the method used to extract the proper words at the proper levels from the data statements.

As soon as the LEN value has been extracted from B$, there is a branch to line 290, where another loop is entered. This one counts from 1 to the value of L. Let's assume that we are working with the word CAT. Therefore, L will be equal to 3. This loop will then make three passes. Line 300 fills the string array (A$). In this case, only three of the available ten array elements will be used, since there are only three letters in CAT. The SEG$ function is used to extract each letter (in order) from the chosen word, which has been assigned to B$. During the first pass of the loop, the letter at the number 1 position in B$ will be assigned to A$(X) or A$(1). This is the letter C. During the next cycle of the loop, A$(2) is opened and the SEG$ function extracts the second letter from B$. This is the letter A. During the third and final pass, the last letter of the word will be assigned to A$(3). The loop times out at this point, and the screen is cleared by the Call Clear subprogram found in line 320. At this point, the string array A$ contains the letters C, A, and T at positions 1, 2, and 3. The remaining seven elements are not assigned and therefore are equal to 0.

At this point, it's time to scramble the letters of the word CAT. To begin with, the randomize statement is used in line 330, and in line 340, the variable RN is assigned the integer value of RND*L + 1. You will remember that L is equal to the number of letters in the word (CAT). Therefore, RN will be equal to either 1, 2, or 3. Line 350 tests for the occurrence of 0 at any of the array elements that will be returned in the future. More on this later. Line 360 prints the letter which is found at A$(RN). In other words, if the random number returned in line 340 is a 3, line 360 will print A$(3), which is the letter T. Line 370 then reassigns A$(RN) (in this case, A$3)) to a value of 0. What is being done here involves removing the letter from the array and placing a 0 in instead. Line 380 is a simple count routine that steps the value of T by 1 each time the line is executed. Line 290 tests for a condition of T being equal to L, which brings about a branch outside of this program loop. The loop is actually started in line 400 with the GOTO branch in line 340. The loop, then, is composed of lines 340 through 400.

Let's assume that the letter T has been extracted from the third element of the array and that this element has now been assigned to the string value 0. It is much easier to understand how this process works when discussing a second pass of the loop. Since we don't know what random number will be returned to variable RN, it is necessary to reassign the element value once it has been read. When the T was extracted from A$(3), the A$(3) position was reassigned to 0. Let's assume that during the next pass, the same number is returned to RN. Line 350 tests for this situation. In this case (assuming that RN is equal to 3), line 350 tells the machine that if A$(3) is equal to 0 (which it now is), then go back to line 340 for another random number. Line 350 will not allow execution to continue past this point until RN is a number that represents an array position that has not yet been read.

Assume further that the next value returned to RN is 2. Line 350 makes certain the value of A$(2) is not 0, and line 360 then prints this value on the screen (the letter A) to the right of the first letter printed (T). The second array position is reassigned a string value of 0, and T is incremented by 1. At present, the value of T (2) is not equal to the value of L (3), so line 400 branches back to line 300 and another random number is output. Assuming the sequence previously discussed, the only number that is left and will be passed through line 350 is the number 1. Line 350 will continue to branch back to line 340 until RN is equal to 1. When a 1 is finally output, the letter at element position A$(1) or C is printed on the screen. Line 380 increments T by 1, and line 390 discovers that T is now equal to L, so the loop is exited by the branch to line 410. The word that now appears on the screen is TAC. Of course, the random number generator output determined the exact order. It could just as easily have been ATC, TCA, or even CAT. Occasionally, the odds are with you, and the word will appear in unscrambled form. This is quite a rare occurrence at any but the beginner difficulty level.

Lines 410 through 450 contain print statements which simply position the scrambled word near the center of the screen. Line 450 prints a screen prompt asking you to guess the word. The input statement in line 480 allows you to type in your guess, and of course, you have as long as you'd like to study the scrambled word on the screen in an effort to come up with the correct answer.

As soon as you've input your guess and pressed the enter key, line 490 clears the screen and line 500 tests the two strings to see if they are equal. The string variable C$ represents your answer, while string variable B$ is the word itself (in unscrambled form), which was read from the data statement. If the two are equal, there is a branch to line 510, which prints the correct word on the screen, followed by a message telling you that your guess was correct. Line 520 is another count routine. The variable SC is incremented by 1 each time a correct answer is input. There is then a branch to line 570; which allows you to receive another scrambled word by pressing the enter key.

On the other hand, if C$ is not equal to B$, the branch is to line 540, which prints "WRONG!!!" on the screen. In line 550, variable SW is incremented by 1. This is the count routine that is accessed whenever an incorrect answer is input. Line 560 then displays the correct word on the screen.

The print statements in lines 570 through 600 again provide proper spacing on the screen for a new display. Line 610 displays a prompt on the screen telling you to press the enter key to continue play. Line 620 accepts this input, while line 630 returns variable T to its original value of 0. There is then a branch to line 240.

This line needs a little more explanation, because it begins another for-next loop similar to the one previously discussed, which began in line 170. You will remember that this loop is the one which counted from 1 to the difficulty factor number. However, it is necessary to use this loop only once, since forever after, only every third word will be accessed from the data statement lines. This applies regardless of the difficulty factor which was initially chosen. The loop, which is begun in line 240, always makes three passes. The read statement in line 250 pulls a word from the data statement each time it is accessed, but only the word pulled during the third pass is evaluated. In other words, the first two words read during the first two passes are simply ignored.

Looking at the end of line 750, you will notice three identical words, each of which is END. This word will be encountered when all of the data words have been exhausted. The word END is included three times to terminate any of the three difficulty factor sequences. Line 260 tests for a condition of B$ being equal to END. When this condition is true, there is a branch to line 760, which clears the screen. The screen then displays a message indicating that the game is over and showing the player's score. Scoring is handled in line 800 and 810, which use variables SC (correct answers) and SW (incorrect answers) with print statement prompts. At this point, the program is ended.

At the beginning of this discussion, it was stated that the program was practically unlimited. This is due to the fact that it can be completely altered by simply changing the words in the data statements. This should take only twenty minutes or so and should be done after several plays, or when you begin to memorize the words which it contains. Naturally, this program should be set up by someone who is not going to be an actual player, since typing in the words yourself gives you an unfair advantage. There is no limitation (practically speaking) on the length of any word, although if there are more than ten letters, it will be necessary to change the DIM statement in line 130 to reflect this. All other program lines may remain the same.

Letter Confusion is a delightful game which combines competition with education. The winner is the one who gets the highest number of correct answers, although you can play against yourself, so to speak, by trying to achieve the highest score possible. The word list is necessarily long to provide the most enjoyment. Also, once you have finished a single run, you can usually start the program all over again, since most of the correct answers will have been forgotten. You can also start out with the lowest difficulty factor, move to intermediate on the second go-around, and finally the expert level when the program is run for the third time. This can triple the enjoyment without having to resort to reprogramming.

If you want to add more words in addition to those already included in the data statements, simply add more data lines. You should first remove the three END words from the end of line 750 and then add as many additional data statements and words as you desire. Remember to input these words in groups of three, as was previously discussed, and to terminate the last data statement line with the triple END sequence. It is quite easy to expand the program to include hundreds of different words, which can be scrambled in thousands of different ways by this interesting program.

MATCHGAME

Matchgame is not really a game, but it can be used to randomly match up partners for true games. For example, it can be used to match up sides for a touch football game. The way it is written here, it's really designed as a dating game, in that it allows you to input the names of up to 10 boys and then the names of a like number of girls. When all names have been input, partners will be randomly matched.

The program is extremely simple. Two arrays are established in lines 130 and 140, each of which can hold a maximum of ten elements. The randomize statement in line 150 reseeds the random number generator. Line 170 includes an input statement that allows you to input the number of couples involved. This number is assigned to the variable C. The for-next loop that is begun in line 190 counts from 1 to the value of C, and during each cycle, you are asked to input the name of a boy. The screen is cleared after each name is input. When this loop times out, another one is begun in line 230, which goes through the same number of cycles, but this time, you are asked to input the name of a girl.

Once all names have been input, the random assignments are made. Starting in line 270, the variable I is assigned a value that is equal to a random number between 1 and a maximum value of C. Line 280 will be skipped over for the time being in this discussion. Line 290 prints the element from A$ that is at the position identified by the random number I. Line 300 then reassigns this position, making it equal to the number 0. This portion of the

Listing 12. Matchgame

```
100  REM MATCHGAME
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES 2/11
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  DIM A$(10)
140  DIM B$(10)
150  RANDOMIZE
160  CALL CLEAR
170  INPUT "HOW MANY COUPLES?
":C
180  CALL CLEAR
190  FOR X= 1 TO C
200  INPUT "NAME OF BOY?":B$(
X)
210  CALL CLEAR
220  NEXT X
230  FOR X=1 TO C
240  INPUT "NAME OF GIRL?":A$
(X)
250  CALL CLEAR
260  NEXT X
270  I=INT(RND*C)+1
280  IF A$(I)="0" THEN 270
290  PRINT A$(I);
300  A$(I)="0"
310  II=INT(RND*C)+1
320  IF B$(II)="0" THEN 310
330  PRINT TAB(15);B$(II)
340  B$(II)="0"
350  GOTO 270
```

program pulls the name of each girl from the A$ array. Lines 310 through 340 are a repeat of the above routine. In this case, II is the random number and is used to pull boys' names from the B$ array. Line 350 branches back to line 270 after the first two names have been printed to allow more names to be extracted. Line 280 is absolutely necessary, in that it automatically branches back to line 270 in the event that it encounters an array element that is equal to the number 0. This assures that all names are extracted from the array and that no name is printed twice.

You can use this program to match people, objects, or even numbers on a random basis. By changing some of the screen prompts, the program can be tailored to address these different applications.

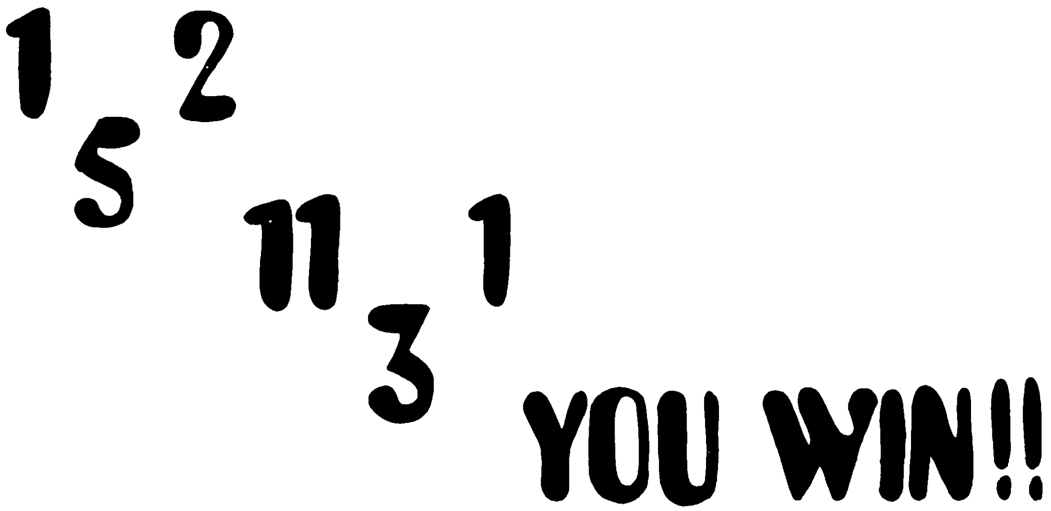
ACEY-DEUCEY

Acey-Deucey is a computerized version of the card game which goes by the same name. The screen will display three numbers. The one in the center must lie between the first and last number shown numerically. In other words, if the first and last numbers are 5 and 10 respectively, any number from 6 to 9 is a winner, since it lies between 5 and 10. Any number less than 5 or greater than 10 is a loser. Likewise, the numbers 5 or 10 are losers.

In the card game, the face cards always counted as 10, the other cards counted as their face values, and the ace could be a 1 or an 11. In this game, only numbers are used, and these will range from 1 to 11. You do not have the option of allowing an 11 to be equal to 1 or 11. This is already written into the program.

The randomize statement is used in line 130 to reseed the random number generator. Lines 150 through 170 assign each of the numerical variables X, Y, and Z a random number between 1 and 11. Lines 180 and 190 display

Listing 13. Acey-Deucey



```
100  REM ACEY-DEUCEY
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES 2/10
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  RANDOMIZE
140  CALL CLEAR
150  X=INT(RND*11)+1
160  Y=INT(RND*11)+1
170  Z=INT(RND*11)+1
180  PRINT "       ";X;"
  ";Y
190  PRINT "            ";Z
200  IF X<Y THEN 210 ELSE 230
210  IF Z>X THEN 220 ELSE 290
220  IF Z<Y THEN 250 ELSE 290
230  IF Z<X THEN 240 ELSE 290
240  IF Z>Y THEN 250 ELSE 290
250  PRINT
260  PRINT
270  PRINT "A WINNER!"
280  GOTO 320
290  PRINT
300  PRINT
310  PRINT "YOU LOSE"
320  PRINT
330  PRINT
340  PRINT
350  PRINT
360  INPUT "PRESS (ENTER)" TO
CONTINUE.":EN$
370  GOTO 140
```

these values in a special format on the screen. The spaces between quotation marks are quite important, so count them carefully. Lines 200 through 240 test for a win. If one is detected, there is a branch to line 250, which prints A WINNER! on the screen. You are then prompted to press the enter key to play again. On the other hand, if a win is not detected, there is a branch to line 290, where YOU LOSE! is printed on the screen.

You can play this game continuously and never be able to predict the outcome. You will probably find, however, that you lose more often using this game than you do when playing the card game equivalent. This is because the odds are different with this game. For example, in cards, if your first number is a 2, you have less chance of getting a 2 again because there are fewer 2s left in the deck. This does not apply with the computerized version, since the computer has a supposedly infinite number of 2s, and for that matter, of any other number, to choose from. In other words, the computer's deck is not limited as is a deck of cards. If a 2 comes up as the first number, the odds on it coming up again are the same as the first time around.

COIN FLIP

One of the simplest random chance games to simulate on a computer is the flip of a coin. From a computer standpoint, the flip involves only two possible random numbers, 1 or 2. This assumes that there are only two possible results when a coin is flipped, one side or the other, better known as heads or tails. Of course, from a realistic standpoint, this is not entirely correct, since there is a very, very slight possibility that the coin could land on its edge and remain in this position. Since this is a million to one shot, we won't even consider it.

Listing 14. Coin Flip

Listing continued.

```
100  REM COIN FLIP
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES 4/21
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  CALL CLEAR
140  RANDOMIZE
150  X=INT(RND*2)+1
160  IF X=1 THEN 170 ELSE 190
170  A$="HEADS"
180  GOTO 200
190  A$="TAILS"
200  PRINT "          ";A$
210  PRINT
220  PRINT
230  PRINT
240  PRINT
250  PRINT
260  PRINT
270  PRINT
280  PRINT
290  PRINT
300  PRINT
310  INPUT "PRESS (ENTER) TO
FLIP AGAIN.":EN$
320  GOTO 130
```

Following the randomize statement in line 140, the variable X is assigned a value which will either be 1 or 2 and which is randomly decided. Line 160 assigns A$ to a value of HEADS (in line 170) when X is equal to 1. Otherwise, A$ is assigned the value of TAILS.

As soon as the random determination has been made, line 200 prints the value of A$ at the center of the screen. The print statements in lines 210 through 300 scroll the message upward so that it rests at the exact center of the screen. Line 310 then allows you to flip again by pressing the enter key. This brings about a branch to line 130, where the program begins again.

COIN FLIP 2

Coin Flip 2 is almost identical to the previous program, except that two hypothetical coins are used. Two random numbers are set up and assigned to X and Y. Again, the range is from 1 to 2, with 1 representing heads and 2 representing tails for both coins. When this program is run, the results of flipping the two coins are displayed on the screen. This allows for even/odd matching, which is a quite common coin flip game.

Program 15. Coin Flip 2

```
100  REM COIN FLIP 2
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES 4/21
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  CALL CLEAR
140  RANDOMIZE
150  X=INT(RND*2)+1
160  Y=INT(RND*2)+1
170  IF X=1 THEN 180 ELSE 200
180  A$="HEADS"
```

Listing continued.

```
190  GOTO 210
200  A$="TAILS"
210  IF Y=1 THEN 220 ELSE 240
220  B$="HEADS"
230  GOTO 250
240  B$="TAILS"
250  PRINT "     ";A$;"     "
;B$
260  PRINT
270  PRINT
280  PRINT
290  PRINT
300  PRINT
310  PRINT
320  PRINT
330  PRINT
340  PRINT
350  PRINT
360  PRINT
370  PRINT
380  INPUT "PRESS (ENTER) TO
FLIP AGAIN.":EN$
390  GOTO 130
```

HIDDEN WORDS

Hidden Words is a computer-assisted game that also involves pencil and paper. Most readers will remember the word games they played as children (and sometimes as adults) that involved a long conglomeration of letters that the players used to make words. Any letter that was found in that conglomeration could be used to form part of a word. The person who made the most words from the string was the winner.

This program randomly outputs a conglomeration of letters with which players can attempt to form words. Each output from the

Listing 16. Hidden Words

```
100  REM HIDDEN WORDS
110  REM COPYRIGHT  FREDERICK
HOLTZ AND ASSOCIATES 4/22/83
120  REM PROGRAM RUNS IN TI-BASIC
130  DIM A$(26)
140  CALL CLEAR
150  FOR X=65 TO 90
160  A$(X-64)=CHR$(X)
170  NEXT X
180  RANDOMIZE
190  CALL CLEAR
200  FOR Z=1 TO 10
210  Y=INT(RND*26)+1
220  PRINT A$(Y);
230  NEXT Z
240  PRINT
250  PRINT
260  PRINT
270  PRINT
280  PRINT
290  PRINT
300  PRINT
310  PRINT
320  PRINT
330  PRINT
340  PRINT
350  PRINT
360  PRINT
370  PRINT
380  INPUT "PRESS (ENTER) FOR
NEW LETTERS.":ER$
390  GOTO 190
```

computer will consist of ten letters. When you press the enter key again, a new string of letters is output.

Line 130 sets up an array which contains 26 elements. The loop found in lines 150 through 170 assigns the letters of the alphabet to each of the 26 array positions. In this case, the letters are ASCII characters 65 through 90, which are the capital letters A through Z in the ASCII format of the TI-99/4A (and in nearly every other computer as well).

The randomize statement is used in line 180, and line 200 begins a loop which counts from 1 to 10. With each cycle, Y is assigned a new random number and this number pulls the letter from the A$ appropriate position in the array. Since we don't care if letters are repeated, it's not necessary to reassign the array position value, as was done with the random match program discussed previously. After the loop has completed ten cycles, it times out and the print statements contained in lines 240 through 370 center the word on the display screen. Lines 380 and 390 allow for a new series of letters to be printed when the enter key is pressed.

MATHEMATICAL PROGRESSION

Mathematical Progression is an arithmetic game that can be enjoyed by persons of all ages. While the program shown here may be used as is, it is presented in order to give the reader a program format and an idea for writing his own program.

This program contains three different mathematical formulas held within for-next loops. The loops begin at lines 140, 210, and 280, respectively. Each of these loops causes a series of numbers to be displayed on the screen. It is the player's responsibility to determine the relationship of these numbers and then to input the next number that would logically follow in the string. For example, if the computer displayed 1 2 3 4 5 6, the next logical number would obviously be 7, since the numbers are pressing upward in steps of 1. Depending on the mathematical formula used to determine the numbers, the program's difficulty factor can be high or low.

For example, consider the string which will be generated by lines 140 through 170. B will be equal to the loop value divided by 8. Each time the loop cycles, the value of A divided by 8 is displayed on the screen. When the loop times out, line 180 assigns a value to BB which is equal to 11 divided by 8, the next logical number in this string. There is then a branch to a routine beginning in line 390, which adds two blank lines to separate the prompts to be printed from the numbers already on the screen and also steps the value of R (the number of attempts) by 1 (see line 410). The player is then prompted to input the next number in the sequence. This input is assigned to the variable AA in line 440. Line 450 compares this input with the correct answer, which was assigned to BB. If the two match, there is a branch to line 560, which informs you that your answer is correct. If the two are not equal, line 460 is executed and causes the screen to display WRONG ANSWER!!. Following this, the value of W in line 470 is stepped by 1. When you press the enter key again, the return statement is executed and there is a branch back to line 200, where the screen is cleared and a new problem is encountered.

You can custom-tailor this program by inserting as many new problems and for-next

Listing 17. Mathematical Progression

```
100  REM MATHEMATICAL PROGRES
SION
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES 4/20
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  CALL CLEAR
140  FOR A=1 TO 10
150  B=A/8
160  PRINT B;
170  NEXT A
180  BB=11/8
190  GOSUB 390
200  CALL CLEAR
210  FOR A=20 TO 35
220  B=A*.5
230  PRINT B;
240  NEXT A
250  BB=36*.5
260  GOSUB 390
270  CALL CLEAR
280  FOR A=20 TO 36
290  B=SQR(A)
300  PRINT B;
310  NEXT A
320  BB=SQR(37)
330  GOSUB 390
340  CALL CLEAR
350  PRINT "THE GAME IS ENDED "
360  PRINT "YOU HAD";W;"WRONG
 ANSWERS"
370  PRINT "OUT OF";R;"ATTEMP
TS."
380  END
390  PRINT
400  PRINT
410  R=R+1
420  PRINT "WHAT IS THE NEXT
NUMBER"
430  PRINT "IN THIS SEQUENCE?
"
440  INPUT AA
450  IF AA=BB THEN 560
460  PRINT "WRONG ANSWER!!"
470  W=W+1
480  PRINT "DO YOU WANT TO TR
Y AGAIN(Y/N)?"
490  INPUT AG$
500  IF AG$="Y" THEN 390
510  PRINT "THE CORRECT ANSWE
R IS";BB
520  PRINT
530  PRINT
540  INPUT "PRESS ENTER":AS$
550  RETURN
560  PRINT "THAT IS CORRECT!!
"
570  GOTO 540
```

loops as you want. This is the area of the program currently filled by the statements in lines 140 through 330. At the end of each problem sequence, there is a GOSUB to the subroutine that allows you to input an answer and also keeps score. At the end of the program, the number of attempts you made to answer the problems and the number of wrong answers is displayed. This is a basic program that can be lengthened tremendously to be developed into an extremely complex and educational mathematics drill.

MATH GAME

Another mathematics game that is enjoyed by a wide range of individuals involves the display of math problems on the screen. Naturally, the player is supposed to come up with the correct answer. This version is fairly simple, but by changing the random value range in lines 340 and 350, the difficulty level can be increased or decreased. This program displays addition, subtraction, multiplication, and division problems on the screen.

The numbers that are added, subtracted, multiplied, or divided are random in nature and are determined by lines 340 and 350. In this

Listing 18. Math Game

```
100  REM MATH GAME
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES 4/23
/83
120  REM PROGRAM RUNS IN TI-B
ASIC
130  RANDOMIZE
140  CALL CLEAR
150  PRINT "THIS IS A MATHEMA
TICAL"
160  PRINT
170  PRINT "QUIZ GAME.  ANY NU
MBER"
180  PRINT
190  PRINT "OF MATHEMATICAL P
ROBLEMS"
200  PRINT
210  PRINT "CAN BE DISPLAYED
ON THE"
220  PRINT
230  PRINT "SCREEN. EACH TIME
 A NEW ONE"
240  PRINT
250  PRINT "APPEARS, YOU ARE
TO INPUT"
260  PRINT
270  PRINT "AN ANSWER. YOU WI
LL BE"
280  PRINT
290  PRINT "INFORMED AS TO IT
S ACCURACY."
300  PRINT
310  PRINT
320  INPUT "PRESS (ENTER) TO
CONTINUE.":QW$
330  CALL CLEAR
340  X=INT(RND*10)+1
350  Y=INT(RND*10)+1
360  Z=INT(RND*4)+1
370  IF Z=1 THEN 380 ELSE 410
380  S$="+"
390  ANS=X+Y
400  GOTO 510
410  IF Z=2 THEN 420 ELSE 450
420  S$="-"
430  ANS=X-Y
440  GOTO 510
450  IF Z=3 THEN 460 ELSE 490
460  S$="*"
470  ANS=X*Y
480  GOTO 510
490  S$="/"
500  ANS=X/Y
510  PRINT X;S$;Y;"="
520  INPUT QR
530  IF QR=ANS THEN 540 ELSE
610
540  PRINT
```

Listing continued.

```
550  PRINT
560  PRINT "THAT IS CORRECT"
570  PRINT
580  PRINT
590  INPUT "PRESS (ENTER) TO
CONTINUE.":FR$
600  GOTO 330
610  PRINT "THAT IS A WRONG A
NSWER!!"
620  PRINT
630  PRINT "THE CORRECT ANSWER
IS";ANS
640  GOTO 570
```

program, the numbers range from 1 to 10 and are assigned to variables X and Y. Variable Z is assigned a random number of from 1 to 4, and this number determines the mathematical function (addition, subtraction, multiplication, division). For example, in line 370, if Z is equal to 1, there is a branch to line 380, where S$ is equal to a plus sign (+). Line 390 then assigns the variable ANS to be equal to the sum of X and Y. If Z is equal to 2, Y is subtracted from X. The number 3 means that X will be multiplied by Y. If Z is equal to 4, then the division function is accessed. In each case, ANS is assigned the value of the function results of X and Y.

Line 510 prints the value of X followed by the mathematical function sign, the value of Y and then the equal sign. In one example, the screen might display

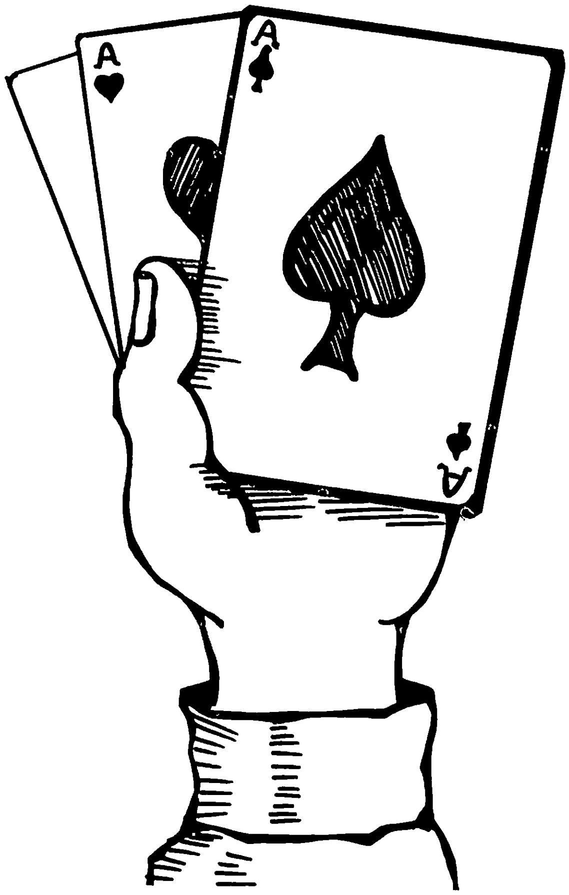
    10 + 8 =

You would then input the value of 18. Line 530 checks for the inputting of a correct value. The variable QR represents the keyboard input, and if it is equal to ANS (the correct answer), there is a branch to line 540, which prints the CORRECT prompt on the screen. If the two are not equal, there is a branch to line 610, which causes the screen to display the fact that a wrong answer has been input. This is followed by the printing of the correct answer. As soon as this sequence is over, you can press the enter key again and a new problem will be displayed on the screen. Unlike the previous program, all problems are determined at random, and you can play all day and all night with this same program and still not be repeating too many problems. Of course, if you allow X or Y to be equal to a random number in the range of from 1 to 20 (instead of 1 to 10), the difficulty factor is increased and there are many more possible number combinations. By decreasing the random number range, the difficulty factor can be lowered for children. You may also wish to omit the multiplication and division aspects and simply leave the addition and subtraction problems. As you can see, this program can be easily tailored to meet the needs of a wide range of individuals and is popular with young and old alike.

BLACKJACK

Since the home computer became popular, the game of Blackjack has been programmed many times and in many ways for each and every model. For our purposes, it would be more appropriate to call it the game of twenty one, since cards are not actually

Listing 19. Blackjack Card Game



```
100  REM BLACKJACK CARD GAME
110  REM TI VERSION COPYRIGHT
120  REM  FREDERICK HOLTZ  AND
ASSOCIATES 3/4/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  RANDOMIZE
160  D=0
170  P=D
180  GOSUB 630
190  D1=C
200  GOSUB 630
210  D2=C
220  GOSUB 700
230  P1=C
240  GOSUB 700
250  P2=C
260  PRINT
270  PRINT "THE DEALER HAS";D
1;"SHOWING"
280  PRINT "YOU HAVE";P1;"AND
 ";P2
290  PRINT "YOUR TOTAL IS";P1
+P2
300  D=D1+D2
310  P=P1+P2
320  IF P=21 THEN 440
330  GOSUB 770
340  IF L=1 THEN 490
350  IF D<=16 THEN 550
360  PRINT "THE DEALER HAS";D
370  PRINT "YOU HAVE ";P
380  IF P>D THEN 420
390  REM
400  PRINT "DEALER WINS!!!"
410  GOTO 890
420  PRINT "YOU ARE THE WINNE
R!!!"
430  GOTO 890
440  PRINT "YOU HAVE BLACKJAC
K"
450  IF D=21 THEN 470
460  GOTO 360
470  PRINT "DEALER ALSO HAS B
LACKJACK--NO WINNER!!!"
480  GOTO 890
```

Listing continued.

```
490  GOSUB 700
500  PRINT "YOUR CARD IS ";C
510  P=P+C
520  PRINT "YOUR TOTAL IS";P
530  IF P>21 THEN 400
540  GOTO 330
550  PRINT "THE DEALER HAS";D
560  GOSUB 630
570  LET D=D+C
580  PRINT "THE DEALER DRAWS
A ";C
590  PRINT "DEALER TOTAL IS " ;D
600  IF D>21 THEN 420
610  IF D< = 16 THEN 560
620  GOTO 360
630  C=INT(RND*11)+1
640  IF C=11 THEN 660
650  GOTO 690
660  IF D+C>21 THEN 680
670  GOTO 690
680  C=1
690  RETURN
700  C=INT(RND*11)+1
710  IF C=11 THEN 730
720  GOTO 760
730  IF P+C>21 THEN 750
740  GOTO 760
750  C=1
760  RETURN
770  PRINT "HIT OR STAY? (H/S)"
780  INPUT Q$
790  IF Q$="H" THEN 830
800  IF A$="S" THEN 860
810  PRINT "TRY AGAIN"
820  GOTO 770
830  L=1
840  CALL CLEAR
850  GOTO 880
860  L=0
870  CALL CLEAR
880  RETURN
890  PRINT
900  PRINT "PRESS (ENTER) TO
PLAY AGAIN."
910  INPUT L$
920  GOTO 140
```

used, having been replaced by the numbers 1 through 11. As in the card game, the idea is to get a number combination that will result in a maximum of 21. If you go over this figure, you're busted. The player with the highest score under 21 is the winner. In this game, there is only one player, at least of the human type. Here, the computer is the dealer and, of course, a tie automatically goes to the dealer.

This program does not use an array to hold 52 different numbers in order to simulate a card deck. Instead, it uses random number generators to output numbers from 1 to 11. Therefore, the odds must be calculated in a different manner than when playing Blackjack. In a standard card deck, there are four of each number and/or face card. Naturally, if the dealer is displaying a queen and you have a queen showing and one face down, there is little likelihood that you will draw the fourth and final queen, or that the dealer will already have it face down. Of course, the other face cards count 10 as well, but there is even less of a chance of you actually getting the last queen. The computer version of this game makes no distinction between the different suits and types of face cards. It represents them all by the number 10. However, there is a theoretically infinite number of tens in this game, and the same goes for every other card. Each number is chosen from between 1 and 11 by random chance, so you can never run out of our representative face cards or any other number.

Other than these differences, the game is readily played by anyone who already knows how to play Blackjack. Each time the program is run, you are automatically assigned two numbers and their total is given. You will also be told the value of one number the computer is holding. You may then opt to take a hit or pass. If you go over 21, the screen clears, and the dealer is declared the winner. This program will also test for Blackjack by you or the dealer, or by both at the same time.

Again, this program is a modification of one presented in another TAB publication, but it has been completely rewritten for the TI-99/4A and tested on this same machine. The modifications include getting rid of statements and functions not applicable to TI BASIC, and arranging the display to fit the 32-column TI-99/4A format. In making such conversions, it is often necessary to rewrite many if-then lines. Many of these execute separate statements on the same line, depending on the if-then test. This is not permitted in TI BASIC (it is in Extended BASIC), so an if-then line must always branch to another line.

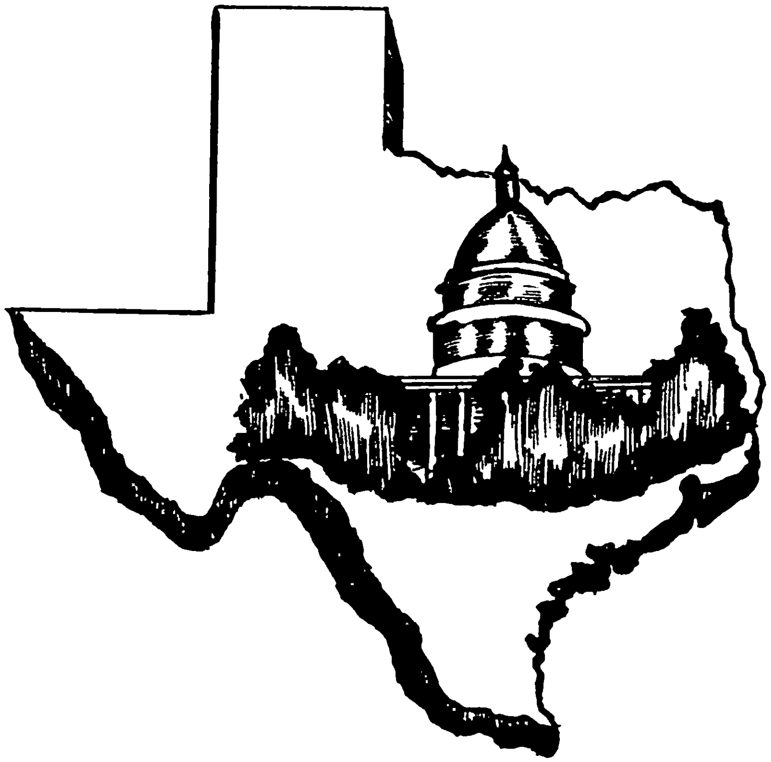
STATES AND CAPITALS

Here's a straightforward program that is a great tutorial for school age children and will also stymy many adults who haven't had to deal with the capitals of various states for many years. The program is extremely simple, but it is long due to the fact that each data statement contains the name of a state, along with its capital. It would have been possible to conserve a fair amount of memory by including several states and their capitals on a single data statement line. However, this method assures easy debugging and makes the program content much more clear.

When the program is first run, line 140 clears the screen and lines 150 and 160 assign the state to A$ and its capital to B$. Line 180 prompts the player to input the capital of the state contained in A$. The player's input is assigned to C$ in line 220. Line 230 tests for a condition of C$ (player's answer) being equal to B$ (correct capital). If the two match, there is a branch to line 240, which clears the screen. Line 250 is a counting routine which steps the variable RA by 1 each time a correct answer has been input. Following this, line 260 prints the capital, followed by the CORRECT message. The player is then asked if he wishes to play again, and if so, there is an eventual branch to line 150, where the next state and capital are read from the data statement lines.

In the event that an incorrect answer is input, line 230 branches to line 380, where you are told that a wrong answer has been given. The variable WA stands for wrong answer and is stepped by 1 each time an incorrect answer is input. When a wrong answer occurs, the player is asked if he wishes to try again. If a Y is input, the state name is displayed again and another guess may be attempted. When the player simply cannot come up with the answer, he will enter an N at line 400. This brings about a branch to line 430, which clears the screen, and then to lines 440 and 450, which print the correct answer on the screen. The player may

Listing 20. States and Capitals



```
100  REM STATES AND CAPITALS
110  REM COPYRIGHT  FREDERICK
HOLTZ  AND ASSOCIATES
120  REM 4/29/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  READ A$
160  READ B$
170  CALL CLEAR
180  PRINT "WHAT IS THE CAPITAL
OF"
190  PRINT A$
200  PRINT
210  PRINT
220  INPUT C$
230  IF C$=B$ THEN 240 ELSE 380
240  CALL CLEAR
250  RA=RA+1
260  PRINT C$;" IS CORRECT!!"
270  PRINT
280  PRINT
290  PRINT
300  PRINT
310  PRINT "PLAY AGAIN?(Y/N)"
320  INPUT ER$
330  IF ER$="Y" THEN 360
340  IF ER$="N" THEN 990
```

Listing continued.

```
350  GOTO 310
360  CALL CLEAR
370  GOTO 150
380  PRINT "WRONG!! TRY AGAIN
?(Y/N)"
390  WA=WA+1
400  INPUT WR$
410  IF WR$="Y" THEN 420 ELSE
 430
420  GOTO 170
430  CALL CLEAR
440  PRINT "THE CORRECT ANSWE
R IS "
450  PRINT B$
460  PRINT
470  PRINT
480  GOTO 270
490  DATA ALABAMA,MONTGOMERY
500  DATA ALASKA,JUNEAU
510  DATA ARIZONA,PHOENIX
520  DATA ARKANSAS,LITTLE ROC
K
530  DATA CALIFORNIA,SACRAMENTO
540  DATA COLORADO,DENVER
550  DATA CONNECTICUT,HARTFOR
D
560  DATA DELAWARE,DOVER
570  DATA FLORIDA,TALLAHASSEE
580  DATA GEORGIA,ATLANTA
590  DATA HAWAII,HONOLULU
600  DATA IDAHO,BOISE
610  DATA ILLINOIS,SPRINGFIEL
D
620  DATA INDIANA,INDIANAPOLI
S
630  DATA IOWA,DES MOINES
640  DATA KANSAS,TOPEKA
650  DATA KENTUCKY,FRANKFORT
660  DATA LOUISIANA,BATON ROU
GE
670  DATA MAINE,AUGUSTA
680  DATA MARYLAND,ANNAPOLIS
690  DATA MASSACHUSETTS,BOSTO
N
700  DATA MICHIGAN,LANSING
710  DATA MINNESOTA,SAINT PAU
L
720  DATA MISSISSIPPI,JACKSON
730  DATA MISSOURI,JEFFERSON
CITY
740  DATA MONTANA,HELENA
750  DATA NEBRASKA,LINCOLN
760  DATA NEVADA,CARSON CITY
770  DATA NEW HAMPSHIRE,CONCO
RD
780  DATA NEW JERSEY,TRENTON
790  DATA NEW MEXICO,SANTA FE
800  DATA NEW YORK,ALBANY
810  DATA NORTH CAROLINA,RALE
IGH
820  DATA NORTH DAKOTA, BISMAR
CK
830  DATA OHIO,COLUMBUS
840  DATA OKLAHOMA,OKLAHOMA C
ITY
850  DATA OREGON,SALEM
860  DATA PENNSYLVANIA,HARRIS
BURG
870  DATA RHODE ISLAND,PROVID
ENCE
880  DATA SOUTH CAROLINA,COLU
MBIA
890  DATA SOUTH DAKOTA, PIERRE
900  DATA TENNESSEE,NASHVILLE
```

Listing continued.

```
910  DATA TEXAS,AUSTIN
920  DATA UTAH,SALT LAKE CITY
930  DATA VERMONT,MONTPELIER
940  DATA VIRGINIA,RICHMOND
950  DATA WASHINGTON,OLYMPIA
960  DATA WEST VIRGINIA,CHARL
ESTON
970  DATA WISCONSIN,MADISON
980  DATA WYOMING,CHEYENNE
990  CALL CLEAR
1000  PRINT "YOU HAD";RA;"COR
RECT AND"
1010  PRINT WA;"INCORRECT ANS
WERS."
1020  PRINT
1030  PRINT
1040  END
```

exit the program after each guess, and when this is done, there is a branch to line 990, where the screen is cleared and you are given your score in terms of right and wrong answers.

ONE MILLION WORDS

Here is a program that I just happened across while writing a more complex one which will be presented next. The name of this program is quite applicable in that it can display one million words or more if you play it long enough. This is a hidden word puzzle with little or no logic behind it. Chances are, you've seen many of these puzzles in various magazines. A large number of letters is printed horizontally and vertically on the page with certain words occurring horizontally, verti-

Listing 21. One Million Words

```
100  REM ONE MILLION WORDS
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 3/15/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  DIM A(26)
150  CALL CLEAR
160  FOR X=1 TO 26
170  A(X)=X+64
180  NEXT X
190  RANDOMIZE
200  CALL CLEAR
210  RN=INT(RND*26)+1
220  CV-CV+1
230  PRINT CHR$(A(RN));
240  IF CV<=28 THEN 260
250  CV=0
260  GH=GH+1
270  IF GH=616 THEN 290
280  GOTO 210
290  PRINT
300  PRINT
310  INPUT "PRESS ENTER FOR N
EW PUZZLE.":ER$
320  GH=0
330  CV=0
340  GOTO 200
```

cally, diagonally, and in reverse order. Most of the letters are simply a jumble, but some of them line up to form the words that must be extracted. This is usually done by circling the word.

This program works along similar lines, except all letters are generated on a random basis. No specific English words are actually programmed for, although they will certainly occur during every program run. A total of 616 letters will be generated on each screen, and it is then up to the players (two or more) to extract as many words as possible from this maze. Although no true words are purposely programmed, I guarantee that you will be surprised at some of the words that crop up at random.

WORD PUZZLE PLUS

This program is the one that sprang from the preceding word puzzle routine. It is handled in almost exactly the same manner, except a number of words are purposely programmed into the puzzle.

Listing 22. Word Puzzle Plus

```
100  REM WORD PUZZLE PLUS
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 4/1/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  DIM A(26)
150  CALL CLEAR
160  FOR X=1 TO 26
170  A(X)=X+64
180  NEXT X
190  RANDOMIZE
200  CALL CLEAR
210  RN=INT(RND*26)+1
220  CV=CV+1
230  PRINT CHR$(A(RN));
240  IF CV<=28 THEN 260
250  CV=0
260  GH=GH+1
270  IF GH=616 THEN 290
280  GOTO 210
290  READ A$
300  IF A$="END END" THEN 310
 ELSE 340
310  CALL CLEAR
320  PRINT "NO MORE PUZZLES!!
"
330  END
340  IF A$="END" THEN 350 ELS
E 410
```

Listing continued.

```
350  PRINT
360  PRINT
370  INPUT "PRESS ENTER FOR N
EW PUZZLE.":ER$
380  GH=0
390  CV=0
400  GOTO 200
410  LV=LEN(A$)
420  I=INT(RND*27)+2
430  II=INT(RND*19)+3
440  IF I+LV > 28 THEN 420
450  FOR T=1 TO LV
460  Q$=SEG$(A$,T,1)
470  W=ASC(Q$)
480  CALL HCHAR(II,I+T,W)
490  NEXT T
500  GOTO 290
510  DATA BASIC,LOGIC,KEY,DAT
A,INPUT,OUTPUT,MEMORY,HARDWA
RE,SOFTWARE,MATRIX,END
520  DATA TRANSISTOR,CHIP,FLO
WCHART,BIT,BYTE,PRINTER,MODE
M,DISK,FLOPPY,TRACK,END
530  DATA HANDSHAKE,PARITY,DA
TA,LOGIC,MONITOR,DISKETTE,RG
B,JACK,MODULATOR,FANOUT,END
540  DATA DRIVE,CARD,CHIP,GIG
O,CMOS,CIRCUIT,CHASSIS,DATAB
ASE,END
550  DATA END END
```

These words are contained in the data statements found in lines 510 through 540. The puzzle discussed previously is printed on the screen, and the data statement words are written over the other letters at random locations. Sometimes, one data statement word will cover up another, so all of the words contained in the data statement lines may not appear in any given puzzle. Each line contains the words that will be contained in one puzzle, and as written, there are enough data statement lines to produce four separate puzzles. You can produce many others by simply adding more data statement lines between current program lines 540 and 550. Each data statement line is terminated with the word "END". This serves as an indication that no further data statement words are to be added. When one puzzle is completed, you simply press the enter key to get a completely different puzzle. After four puzzles have been worked, the last data statement line, which contains the word "END" twice, is encountered. This is a sign to the computer that there are no more data statement lines and the program is to be terminated. Lines 410 through 490 read the number of letters in each data statement word and assign them to various random positions in the puzzle. These lines assure that no words will be started at a point near the right-hand side where there are not enough additional screen column positions to complete the word. Theoretically, you may input words of any length up to 28 letters via the data statement lines. The puzzle is a total of 28 columns wide.

NUMBERS ALIGNMENT

This program is seen in many different forms. Most use the numbers 1 through 5, but this one is expanded to include numbers 1 through 9. The numbers will be displayed in random order. The player's job is to align these numbers in sequential order (1-9).

When the program is first run, you will see a random pattern, such as

9 1 2 4 7 3 5 6 8

When this display is complete, a prompt will appear asking WHICH NUMBER. The number you select will completely rearrange this pattern. Let's assume that you select 4, given the display shown above. The screen will clear, and a new pattern, which is no longer chosen at random but determined by the previous random pattern and the number you selected will be generated. The 4 that you selected will result in a display of

4 2 1 9 7 3 5 6 8

You can see here that the number selected automatically goes to the front of the string, and all numbers that previously preceded it now follow it. All numbers to the left of the one chosen are left in the same order. I will not explain this program further, because it will give you a clue as to how it may be deciphered. Here is one of the few programs included in this book that depends on intelligence and does not give the programmer a playing advantage. In other words, by inputting these program lines, you will still have the same amount of difficulty in solving each sequence as someone with no programming skills at all.

While I won't give you a hint, I will say that any sequence displayed by the computer can be rearranged in sequential order. As soon as you get the idea, you will be able to solve any sequence quite easily. The simple trick to a quick solution may take a half hour or more, but most players will eventually catch on. From there on out, it's a matter of two players with similar experience trying to outdo each other. Each time the sequence is solved, the computer will tell you how many moves it took you to arrive at the solution.

Listing 23. Numbers Alignment

```
100  REM NUMBERS ALIGNMENT
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 4/28/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  PRINT "THIS IS A GAME OF
 SKILL"
160  PRINT
170  PRINT "CALLED 'ALIGNMENT
'. TO WIN"
180  PRINT
190  PRINT "YOU MUST ARRANGE
THE NUMBERS"
200  PRINT
210  PRINT "DISPLAYED ON THE
SCREEN"
220  PRINT
230  PRINT "IN SEQUENTIAL ORD
ER. THE"
240  PRINT
250  PRINT "FIRST GROUPING OF
 NUMBERS IS"
```

Listing continued.

```
260  PRINT
270  PRINT "CHOSEN AT RANDOM.
 YOU WILL"
280  PRINT
290  FOR X=1 TO 1000
300  NEXT X
310  PRINT "DECIDE THE FUTURE
 ORDERS."
320  PRINT
330  PRINT "WHEN YOU INPUT A
NUMBER, IT"
340  PRINT
350  PRINT "AND ALL NUMBERS T
O ITS LEFT"
360  PRINT
370  PRINT "WILL BE DISPLACED
 FROM LEFT"
380  PRINT
390  PRINT "TO RIGHT. THE COM
PUTER WILL"
400  PRINT
410  PRINT "BE CONSTANTLY CHE
CKING FOR A"
420  PRINT
430  PRINT "WIN. IT WILL TELL
 YOU YOUR"
440  PRINT
450  PRINT "SCORE WHEN YOU GE
T AN"
460  PRINT
470  PRINT "ALIGNMENT. GOOD L
UCK!!"
480  PRINT
490  PRINT
500  INPUT "PRESS (ENTER) TO
CONTINUE.":ER$
510  CALL CLEAR
520  DIM A(10)
530  DIM B(10)
540  RANDOMIZE
550  FOR X=1 TO 9
560  A(X)=X
570  NEXT X
580  Q=0
590  R=0
600  Y=INT(RND*9)+1
610  IF A(Y)=0 THEN 600
620  PRINT A(Y);
630  Q=Q+1
640  B(Q)=A(Y)
650  A(Y)=0
660  IF Q=9 THEN 680
670  GOTO 600
680  PRINT
690  PRINT
700  PRINT
710  PRINT
720  PRINT
730  PRINT
740  PRINT "WHAT NUMBER?"
750  INPUT C
760  PRINT
770  PRINT
780  PRINT
790  PRINT
800  PRINT
810  R=R+1
820  IF C=B(R) THEN 840
830  GOTO 810
840  FOR X=R TO 1 STEP -1
850  Z=Z+1
860  PRINT B(X);
870  A(Z)=B(X)
880  NEXT X
```

Listing continued.

```
890  FOR YY=R+1 TO 9
900  A(YY)=B(YY)
910  PRINT B(YY);
920  NEXT YY
930  FOR T=1 TO 9
940  B(T)=A(T)
950  NEXT T
960  NT=NT+1
970  R=0
980  Q=0
990  Z=0
1000  FOR XYZ=1 TO 9
1010  IF B(XYZ)<>XYZ THEN 119
0
1020  NEXT XYZ
1030  CALL CLEAR
1040  PRINT "1  2  3  4  5  6
  7  8  9"
1050  FOR TONE=1000 TO 3000 S
TEP 20
1060  CALL SOUND(10,TONE,0)
1070  NEXT TONE
1080  PRINT
1090  PRINT
1100  PRINT
1110  PRINT
1120  PRINT
1130  PRINT
1140  PRINT
1150  PRINT "IT TOOK YOU";NT;
"TRYS!"
1160  PRINT
1170  PRINT
1180  END
1190  GOTO 680
```

HI-DI

This is an original computer game that is similar to Blackjack in some ways, but uses a simulated roll of dice to achieve its purpose. The idea of this game is to get 7 points or less with the roll of a maximum of 3 dice. You may stop after the first roll or roll one or two more in the event that your numbers are low. When you have finished your roll, the computer will begin its roll. In this case, a tie does not go to the computer, but is registered as a tie and another game may be played. If you go over 7, you automatically lose and the computer does not have to roll. This program always assumes that you roll first, and the computer has the advantage in that it will determine how many dice it must roll based upon your final score, assuming it is 7 or less.

Listing 24. Hi-Di

```
100  REM HI-DI
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 4/27/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  PRINT "WELCOME TO THE GA
ME OF "
160  PRINT
170  PRINT "HI-DI. THE OBJECT
 IS TO"
```

Listing continued.

```
180  PRINT
190  PRINT "GET A MAXIMUM OF
7 POINTS"
200 PRINT
210  PRINT "FROM THE ROLL OF
THREE DICE."
220  PRINT
230  PRINT "YOU CAN STOP AFTE
R ANY ROLL"
240  PRINT
250  PRINT "AND IF YOU GO OVE
R 7 POINTS,"
260  PRINT
270  PRINT "YOU LOSE! THE COM
PUTER WILL"
280  PRINT
290  PRINT "PLAY AGAINST YOU
AND WILL"
300  PRINT
310  PRINT "TRY TO GET 7 POIN
TS AS WELL."
320  PRINT
330  PRINT "AT THE END OF EAC
H ROLL SE-"
340  PRINT
350  PRINT "QUENCE, THE SCORE
S WILL"
360  PRINT
370  PRINT "BE DISPLAYED. GOO
D LUCK!!"
380  FOR TR=1 TO 1000
390  NEXT TR
400  INPUT "PRESS (ENTER) TO
BEGIN":ER$
410  CALL CLEAR
420  PRINT "YOUR TURN!!"
430  PRINT
440  PRINT
450  PRINT
460  PRINT
470  RANDOMIZE
480  A=INT(RND*6)+1
490  PRINT A
500  PRINT
510  PRINT
520  PRINT
530  PRINT
540  INPUT "ANOTHER ROLL?(Y/N
)":R$
550  IF R$<>"Y" THEN 690
560  CALL CLEAR
570  B=INT(RND*6)+1
580  PRINT A;B
590  IF A+B>7 THEN 690
600  PRINT
610  PRINT
620  PRINT
630  PRINT
640  INPUT "ANOTHER ROLL?(Y/N
)":R$
650  IF R$<>"Y" THEN 690
660  CALL CLEAR
670  C=INT(RND*6)+1
680  PRINT A;B;C
690  PRINT
700  PRINT
710  PRINT
720  PRINT
730  PRINT "YOUR TOTAL IS";A+
B+C
740  FOR TIM=1 TO 500
750  NEXT TIM
760  PRINT
770  PRINT
```

Listing continued.

```
780  PRINT
790  PRINT
800  IF A+B+C < =7 THEN 900
810  PRINT "THE COMPUTER WINS
!!!"
820  PRINT
830  PRINT
840  PRINT
850  INPUT "PRESS (ENTER) TO
PLAY AGAIN":ER$
860  A=0
870  B=0
880  C=0
890  GOTO 410
900  CALL CLEAR
910  PRINT "COMPUTER'S TURN!!
"
920  PRINT
930  PRINT
940  PRINT
950  PRINT
960  AA=INT(RND*6)+1
970  PRINT AA
980  IF AA > A+B+C THEN 1100
990  FOR TIM=1 TO 500
1000  NEXT TIM
1010  CALL CLEAR
1020  BB=INT(RND*6)+1
1030  PRINT AA;BB
1040  IF AA+BB > =A+B+C THEN 11
00
1050  FOR TIM=1 TO 500
1060  NEXT TIM
1070  CALL CLEAR
1080  CC=INT(RND*6)+1
1090  PRINT AA;BB;CC
1100  PRINT
1110  PRINT
1120  PRINT
1130  PRINT
1140  PRINT "YOUR SCORE WAS:"
;A+B+C
1150  PRINT
1160  PRINT "COMPUTER'S SCORE
 IS:";AA+BB+CC
1170  IF AA+BB+CC < =7 THEN 118
0 ELSE 1380
1180  IF AA+BB+CC<A+B+C THEN
1380
1190  IF AA+BB+CC=A+B+C THEN
1200 ELSE 1290
1200  PRINT
1210  PRINT
1220  PRINT
1230  PRINT
1240  PRINT "TIE GAME!!"
1250  PRINT
1260  PRINT
1270  GOTO 1360
1280  PRINT
1290  PRINT
1300  PRINT
1310  PRINT
1320  PRINT "COMPUTER WINS!!"
1330  PRINT
1340  PRINT
1350  PRINT
1360  INPUT "PRESS (ENTER) TO
 PLAY AGAIN.":ER$
1370  GOTO 1430
1380  PRINT
1390  PRINT
1400  PRINT
1410  PRINT "YOU WIN!!!"
1420  GOTO 1330
1430  A=0
```

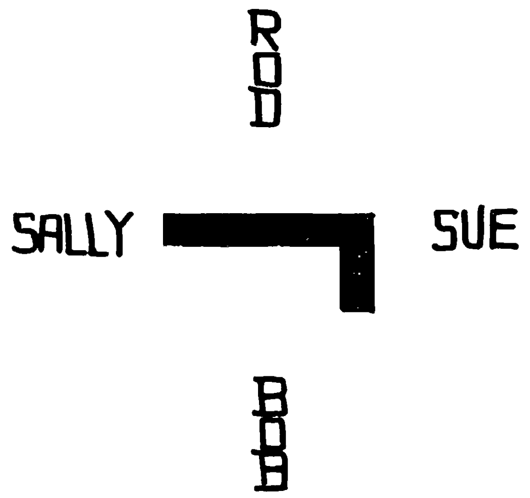
Listing continued.

```
1440  B=0                                    1470  BB=0
1450  C=0                                    1480  CC=0
1460  AA=0                                   1490  GOTO 410
```

SPIN THE BOTTLE

Here is the perfect Spin the Bottle game. It is designed to be played by two couples whose names are input at the beginning. Unlike the conventional Spin the Bottle game, the computer determines who will kiss who, and it always matches the immediate participants up on a boy/girl basis. This game uses a bit of on-screen graphics to point to the names of the players who are to kiss. The names of the girls are input first, and these will appear on the screen at the left and right hand sides. The boys' names will appear vertically at the top and bottom. This arrangement forms a crude circle. When you press the enter key, the spin occurs, and a two-headed pointer will indicate the names of those persons who can kiss. Any combination is possible, but it will always be on a boy/girl basis, provided that the names were properly input at the beginning.

Listing 25. Spin the Bottle



```
100  REM SPIN THE BOTTLE
110  REM COPYRIGHT FREDERICK
HOLTZ  AND ASSOCIATES
120  REM 4/28/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  PRINT "WELCOME TO THE GA
ME OF"
160  PRINT
170  PRINT "SPIN THE BOTTLE W
HICH RE-"
180  PRINT
190  PRINT "QUIRES FOUR PLAYE
RS, TWO"
200  PRINT
210  PRINT "GIRLS AND TWO BOY
S. YOU"
220  PRINT
230  PRINT "WILL BE PROMPTED
TO INPUT"
240  PRINT
250  PRINT "THE NAME OF EACH
PLAYER"
```

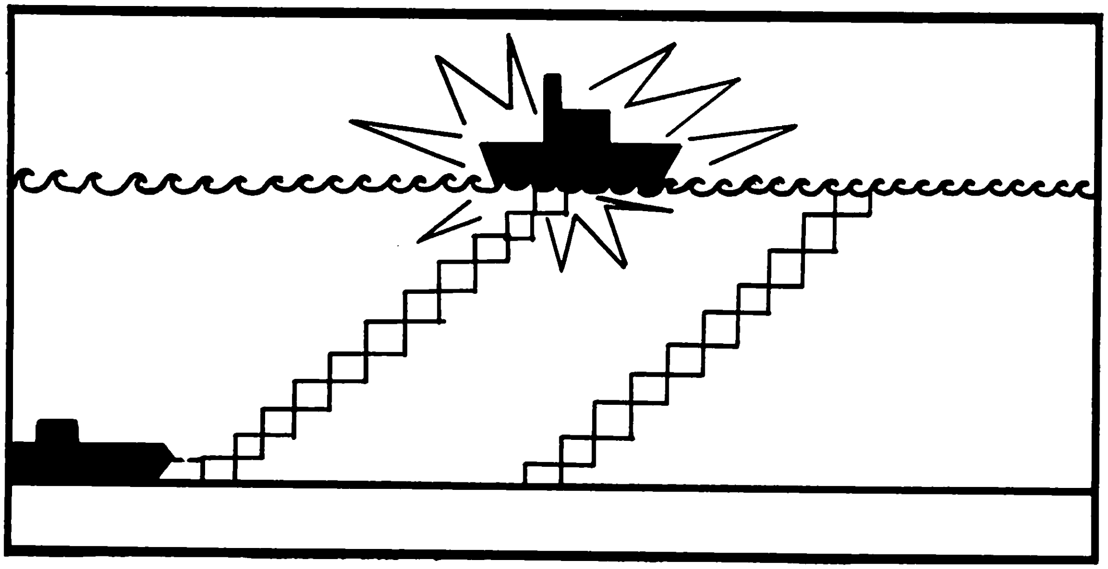
Listing continued.

```
260  PRINT
270  PRINT "AND THEN THE COMP
UTER WILL"
280  PRINT
290  PRINT "SPIN A GRAPHIC BO
TTLE WHICH"
300  PRINT
310  PRINT "WILL POINT TO TWO
 PLAYERS."
320  PRINT
330  PRINT "AT THIS POINT THE
 TWO MAY"
340  PRINT
350  PRINT "KISS. TO SPIN AGA
IN, SIMPLY"
360  PRINT
370  PRINT "PRESS (ENTER)."
380  FOR TD=1 TO 2000
390  NEXT TD
400  RANDOMIZE
410  CALL CLEAR
420  PRINT "FIRST GIRL'S NAME
 IS"
430  INPUT A$
440  CALL CLEAR
450  PRINT "SECOND GIRL'S NAM
E IS"
460  INPUT B$
470  CALL CLEAR
480  PRINT "FIRST BOY'S NAME
IS"
490  INPUT C$
500  CALL CLEAR
510  PRINT "SECOND BOY'S NAME
 IS"
520  INPUT D$
530  CALL CLEAR
540  A=LEN(A$)
550  B=LEN(B$)
560  C=LEN(C$)
570  D=LEN(D$)
580  FOR X=1 TO A
590  AA=ASC(SEG$(A$,X,1))
600  CALL HCHAR(10,X+2,AA)
610  NEXT X
620  FOR X=1 TO B
630  BB=ASC(SEG$(B$,X,1))
640  CALL HCHAR(10,X+23,BB)
650  NEXT X
660  FOR X=1 TO C
670  CC=ASC(SEG$(C$,X,1))
680  CALL HCHAR(X,17,CC)
690  NEXT X
700  FOR X=1 TO D
710  DD=ASC(SEG$(D$,X,1))
720  CALL HCHAR(X+16,17,DD)
730  NEXT X
740  CALL CHAR(33,"FFFFFFFFFF
FFFFFF")
750  SA=INT(RND*2)+1
760  IF SA=1 THEN 770 ELSE 79
0
770  I=7
780  GOTO 800
790  I=10
800  SB=INT(RND*2)+1
810  IF SB=1 THEN 820 ELSE 84
0
820  K=18
830  GOTO 850
840   K=13
850  CALL VCHAR(I,17,33,4)
860  CALL HCHAR(10,K,33,4)
870  INPUT ER$
880  GOTO 530
```

U-BOAT

U-Boat is a simple arcade-like program that graphically displays a blue ocean and a ship moving across its surface. The player assumes the position of U-boat captain and is allowed three torpedoes to try and sink the ship. This game involves a bit of intuitiveness, as the path of the torpedoes is controlled randomly. However, these random paths fall within a certain confined area, and after you've played the game awhile, it is possible to get higher scores based on the torpedo path patterns experienced previously. The torpedoes are fired by pressing any key on the keyboard, and you will see the path of each as it travels through the water toward the ship. The ship must be struck dead center in order to score a kill. When this occurs, you will hear the explosion. Later, the screen will clear and a new ship and ocean will appear, giving you a chance to make another kill. If the ship makes it from one side of the screen to the other, the screen clears and another pass begins. This is a very simple game, but it is quite pleasurable and can provide many hours of enjoyment to both adults and children alike.

Listing 26. U-Boat



```
100  REM U-BOAT
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 4/26/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  PRINT "THIS IS THE GAME OF U-BOAT"
```

Listing continued.

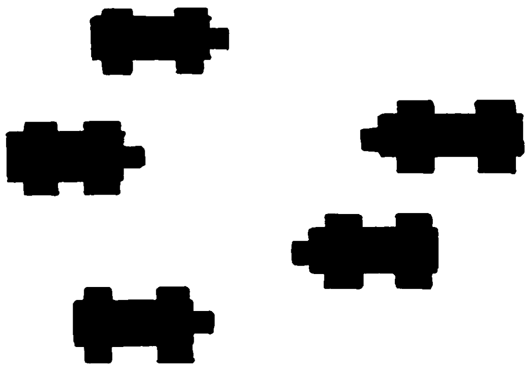
```
160  PRINT
170  PRINT "WHERE YOU ARE THE
 CAPTAIN"
180  PRINT
190  PRINT "OF A SUBMARINE. T
HE OBJECT"
200  PRINT
210  PRINT "OF THE GAME IS TO
 BLOW THE"
220  PRINT
230  PRINT "ENEMY SHIPS OUT O
F THE WATER"
240  PRINT
250  PRINT "AS THEY PASS BY.
YOU GET"
260  PRINT
270  PRINT "THREE TORPEDOS PE
R SHIP."
280  PRINT
290  PRINT "FIRE THEM BY PRES
SING ANY"
300 PRINT
310  PRINT "KEY. GOOD HUNTING
!!!"
320  PRINT
330  PRINT
340  PRINT "PRESS (ENTER) TO
BEGIN."
350  INPUT ENT$
360  CALL CLEAR
370  CALL COLOR(1,7,1)
380  CALL COLOR(5,5,5)
390  CALL SCREEN(4)
400  CALL CHAR(33,"FFFFFFFFFF
FFFFFF")
410  CALL CHAR(34,"0000000000
000000")
420  CALL CHAR(65,"FFFFFFFFFF
FFFFFF")
430  FOR D=11 TO 24
440  CALL HCHAR(D,1,65,32)
450  NEXT D
460  RANDOMIZE
470  FOR X=1 TO 31
480  CALL SOUND(200,-4,2)
490  FR=INT(RND*9)+1
500  CALL HCHAR(10,X,33,2)
510  FOR Y=1 TO 5
520  NEXT Y
530  CALL KEY(0,Z,RD)
540  IF RD=0 THEN 720 ELSE 55
0
550  CT=CT+1
560  IF CT>3 THEN 720
570  CALL SOUND(1000,-8,3)
580  FOR TORP=1 TO 14
590  CALL HCHAR(25-TORP,TORP+
FR,32)
600  FOR TY=1 TO 5
610  NEXT TY
620  NEXT TORP
630  IF TORP+FR=X+1 THEN 640
ELSE 720
640  CALL CLEAR
650  CALL SCREEN(2)
660  CALL SOUND(1500,-7,0)
670  FOR TI=1 TO 250
680  NEXT TI
690  CALL SCREEN(4)
700  CT=0
710  GOTO 430
720  CALL HCHAR(10,X,34,2)
730  FOR Y=1 TO 5
740  NEXT Y
750  NEXT X
760  CALL CLEAR
770  CT=0
780  GOTO 430
```

DEMOLITION DERBY

This is a fairly long and complex program that simulates a demolition derby comprised of four automobiles. The automobiles will be colored red, green, yellow, and blue, and up to four players can each choose a car as their own. When the program is run, these tiny automobiles will appear on the screen for a few seconds, disappear, and then reappear in a different location. If two automobiles come in close contact with each other, there will be a crash sound, and one of them will disappear from the screen. The sequence keeps up until only one automobile is left. The player who has chosen this color is the winner. Sometimes, two automobiles will remain on the screen and never seem to damage each other. If you become tired of waiting, you can always declare the run a draw and start over again. Sometimes, two automobiles will collide without damaging each other severely enough to take one out of the game. Also, there is the possibility that an automobile which was previously damaged (slightly) will suddenly overheat and be taken out.

This game depends solely upon random chance to take automobiles out of the running and determine a winner. All the players have to do is select a color, start the program, and sit back and wait.

Listing 27. Demolition Derby



```
100  REM DEMOLITION DERBY
110  REM COPYRIGHT FREDERICK
HOLTZ AND ASSOCIATES
120  REM 5/2/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  RANDOMIZE
150  CALL CLEAR
160  CALL COLOR(1,7,1)
170  CALL COLOR(2,11,1)
180  CALL COLOR(3,13,1)
190  CALL COLOR(4,5,1)
200  CALL CHAR(33,"0066FEFFFF
FE6600")
210  CALL CHAR(43,"0066FEFFFF
FE6600")
220  CALL CHAR(53,"0066FEFFFF
FE6600")
230  CALL CHAR(63,"0066FEFFFF
FE6600")
240  CALL CHAR(34,"0000000000
000000")
250  RR=33
260  SS=43
270  TT=53
280  UU=63
290  IF WR=1 THEN 320
```

Listing continued.

```
300  A=INT(RND*20)+1
310  AA=INT(RND*20)+5
320  IF WS=1 THEN 350
330  B=INT(RND*20)+1
340  BB=INT(RND*20)+5
350  IF WT=1 THEN 380
360  C=INT(RND*20)+1
370  CC=INT(RND*20)+5
380  IF WU=1 THEN 410
390  D=INT(RND*20)+1
400  DD=INT(RND*20)+5
410  CALL CLEAR
420  IF WR=1 THEN 440
430  CALL HCHAR(A,AA,RR)
440  IF WS=1 THEN 460
450  CALL HCHAR(B,BB,SS)
460  IF WT=1 THEN 480
470  CALL HCHAR(C,CC,TT)
480  IF WU=1 THEN 500
490  CALL HCHAR(D,DD,UU)
500  IF ABS(A-B)< 2 THEN 510 E
LSE 540
510  IF ABS(AA-BB)< 2 THEN 730
 ELSE 540
520  IF ABS(A-C)< 2 THEN 530 E
LSE 540
530  IF ABS(AA-CC)< 2 THEN 840
 ELSE 540
540  IF ABS(A-D)< 2 THEN 550 E
LSE 560
550  IF ABS(AA-DD)< 2 THEN 950
 ELSE 560
560  IF ABS(B-C)< 2 THEN 570 E
LSE 580
570  IF ABS(BB-CC)< 2 THEN 106
0 ELSE 580
580  IF ABS(B-D)< 2 THEN 590 E
LSE 600
590  IF ABS(BB-DD)< 2 THEN 117
0 ELSE 600
600  IF ABS(C-D)< 2 THEN 610 E
LSE 620
610  IF ABS(CC-DD)< 2 THEN 128
0
620  FOR DLAY=1 TO 250
630  NEXT DLAY
640  IF WR=1 THEN 660
650  CALL HCHAR(A,AA,34)
660  IF WS=1 THEN 680
670  CALL HCHAR(B,BB,34)
680  IF WT=1 THEN 700
690  CALL HCHAR(C,CC,34)
700  IF WU=1 THEN 720
710  CALL HCHAR(D,DD,34)
720  GOTO 290
730  CALL SOUND(700,-6,0)
740  FOR DLAY=1 TO 250
750  NEXT DLAY
760  W=INT(RND*2)+1
770  IF W=1 THEN 780 ELSE 810
780  WR=1
790  AA=100
800  GOTO 290
810  WS=1
820  BB=105
830  GOTO 290
840  CALL SOUND(700,-6,0)
850  FOR DLAY=1 TO 250
860  NEXT DLAY
870  W=INT(RND*2)+1
880  IF W=1 THEN 890 ELSE 920
890  WR=1
900  AA=100
910  GOTO 290
920  WT=1
930  CC=110
```

Listing continued.

```
940  GOTO 290
950  CALL SOUND(700,-6,0)
960  FOR DLAY=1 TO 250
970  NEXT DLAY
980  W=INT(RND*2)+1
990  IF W=1 THEN 1000 ELSE 10
30
1000  WR=1
1010  AA=100
1020  GOTO 290
1030  WS=1
1040  BB=105
1050  GOTO 290
1060  CALL SOUND(700,-6,0)
1070  FOR DLAY=1 TO 250
1080  NEXT DLAY
1090  W=INT(RND*2)+1
1100  IF W=1 THEN 1110 ELSE 1
140
1110  WR=1
1120  AA=100
1130  GOTO 290
1140  WU=1
1150  DD=120
1160  GOTO 290
1170  CALL SOUND(700,-6,0)
1180  FOR DLAY=1 TO 250
1190  NEXT DLAY
1200  W=INT(RND*2)+1
1210  IF W=1 THEN 1220 ELSE 1
250
1220  WS=1
1230  BB=105
1240  GOTO 290
1250  WT=1
1260  CC=110
1270  GOTO 290
1280  CALL SOUND(700,-6,0)
1290  FOR DLAY=1 TO 250
1300  NEXT DLAY
1310  W=INT(RND*2)+1
1320  IF W=1 THEN 1330 ELSE 1360
1330  WS=1
1340  BB=105
1350  GOTO 290
1360  WU=1
1370  DD=120
1380  GOTO 290
```

GUNNERY

This program has been around in various forms for a long time and is also known as Artillery, Gunner, Tanks, Azimuth, and many other names. This one was originally written for the TRS-80, but has been modified to run perfectly on the TI-99/4A. It is the culmination of many programs I have seen in the past and includes some special innovations that were included in *The A to Z Book of Computer Games*, by Thomas McIntire, (published by TAB BOOKS Inc.). The mathematics involved are unimportant, and here's how the game is played.

The player assumes the role of gunnery officer and is firing at a target, which is a maximum of 60,000 meters distant. The player's objective is to strike the target by landing a shell within 100 meters of it. This is done by raising or lowering the hypothetical gun barrel. Elevation is input by the player in degrees and fractions of degrees. After you raise or lower the barrel, a shot is fired and the computer spots the shell impact area, telling you whether it was long or short and by how much. By using these readings, you should be able to zero in on the target and blow it up. At the end of the explosion sequence, you will be informed as to the number of shots it took to score the hit. The player with the fewest shots is the winner.

This is a game of skill and intelligence. The actual distance selected at the start of the program is determined by random chance. However, based upon this distance figure, the player must use logic to determine the barrel elevation. As you play this game more and more, you will become more adept at associating certain distances with certain specific elevations. Thus, your scores will improve as the number of shots you take decreases.

Most programs of this type do not include sound effects. However, this one does. Each time you input an elevation and press the enter key, you will hear the sound of the shot being fired; and when you strike the target, an explosion will be heard and the screen will go black for a few seconds. This is a very startling effect, so this type of program will probably hold the player's interest for a longer period than one without such effects.

Listing 28. Gunnery

```
100  REM GUNNERY
110  REM COPYRIGHT FREDERICK
 HOLTZ  AND ASSOCIATES
120  REM 5/1/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  CALL CLEAR
160  PRINT "WELCOME TO THE GA
ME OF GUN-"
170  PRINT
180  PRINT "NERY. YOU ARE THE
 GUNNERY"
190  PRINT
200  PRINT "OFFICER AND ARE T
O FIRE ON"
210  PRINT
220  PRINT "A DISTANT TARGET.
 TO DO"
230  PRINT
240  PRINT "THIS, YOU MUST CO
NSIDER"
250  PRINT
260  PRINT "THE RANGE AND INP
UT THE"
270  PRINT
280  PRINT "PROPER ELEVATION
WHICH"
290  PRINT
300  PRINT "IS OFTEN A DECIMA
L SUCH AS"
310  PRINT
320  PRINT "65.3 FOR INSTANCE
. THE COM-"
330  PRINT
340  PRINT "PUTER WILL TELL Y
OU WHETHER"
350  PRINT
360  PRINT "YOUR SHOT IS LONG
 OR SHORT"
```

Listing continued.

```
370  PRINT
380  PRINT "AND BY HOW MANY M
ETERS."
390  FOR DLAY=1 TO 2000
400  NEXT DLAY
410  CALL CLEAR
420  PRINT "WHEN YOU STRIKE T
HE TARGET,"
430  PRINT
440  PRINT "THE COMPUTER WILL
 TELL YOU"
450  PRINT
460  PRINT "HOW MANY SHOTS YO
U FIRED."
470  PRINT
480  PRINT "GOOD LUCK!!!"
490  PRINT
500  PRINT
510  PRINT
520  PRINT "PRESS (ENTER) TO
CONTINUE."
530  INPUT ER$
540  RANDOMIZE
550  Y=59000-INT(RND*24000)+1
560  CALL CLEAR
570  PRINT "MAXIMUM RANGE IS
60,000"
580  PRINT "METERS. DISTANCE
TO TARGET"
590  PRINT "IS";Y;"METERS."
600  PRINT
610  PRINT "ELEVATION"
620  INPUT E
630  CALL SOUND(250,-6,0)
640  NJ=NJ+1
650  IF E<85 THEN 1040
660  IF E>95 THEN 860
670  CALL CLEAR
680  PRINT "YOUR SHOT WENT ST
RAIGHT UP!"
690  PRINT
700  FOR I=1 TO 10
710  FOR DLAY=1 TO 150
720  NEXT DLAY
730  PRINT TAB(15);"."
740  CALL SOUND(100,1500,0)
750  NEXT I
760  PRINT TAB(13);"BOOM!!"
770  FOR DLAY=1 TO 50
780  NEXT DLAY
790  CALL SOUND(2000,-6,0)
800  CALL SCREEN(1)
810  FOR DLAY=1 TO 100
820  NEXT DLAY
830  CALL SCREEN(8)
840  CALL CLEAR
850  GOTO 610
860  CALL CLEAR
870  PRINT "THAT SHOULD MAKE
YOU A HERO!"
880  PRINT "THAT ROUND MIGHT
HIT MARS!"
890  C=1
900  FOR DLAY=1 TO 750
910  NEXT DLAY
920  GOTO 840
930  CALL CLEAR
940  PRINT TAB(3);"NEWS FLASH
!!!"
950  PRINT
960  PRINT
970  PRINT "MARS STRUCK"
980  PRINT "BY MYSTERY PROJEC
TILE!!"
```

Listing continued.

```
990  PRINT
1000 C=0
1010  RETURN
1020  PRINT "ILLEGAL!!"
1030  GOTO 840
1040  IF C<>1 THEN 1060
1050  GOSUB 930
1060  IF E<1 THEN 1020
1070  E2=2*E/57.2958
1080  J=60000*SIN(E2)
1090  N=Y-J
1100  D=INT(N)
1110  IF ABS(D)< 100 THEN 1140
1120  IF Y-J< 0 THEN 1330
1130  IF Y-J> 0 THEN 1350
1140  CALL CLEAR
1150  CALL SOUND(4000,-6,0)
1160  CALL SCREEN(2)
1170  FOR DLAY=1 TO 850
1180  NEXT DLAY
1190  CALL CLEAR
1200  CALL SCREEN(8)
1210  PRINT "TARGET DESTROYED !!!"
1220  PRINT
1230  PRINT
1240  PRINT
1250  PRINT
1260  PRINT "YOU FIRED";NJ;"R
OUNDS!!"
1270  PRINT
1280  PRINT
1290  PRINT
1300  INPUT "PRESS (ENTER) TO
 CONTINUE":ER$
1310  NJ=0
1320  GOTO 560
1330  PRINT "SHORT BY";ABS(D)
;"YARDS"
1340  GOTO 610
1350  PRINT "LONG BY";ABS(D);
"YARDS"
1360  GOTO 610
```

CRYPTO

Crypto is a decoding game in which the computer substitutes a different letter for each letter in the alphabet. Short sentences or phrases included in data statements within the program are then encoded and displayed on the screen. The player is to figure out the code and then type the decoded message using the keyboard. The computer will then examine the answer given by the player and indicate whether it is right or wrong. If it is incorrect, the correct message will also be displayed.

Code deciphering games have been popular for hundreds of years, and even today syndicated columns that supply code games appear in many American newspapers. This one works very much like the printed games, and the computer is used to come up with an infinite number of coding structures.

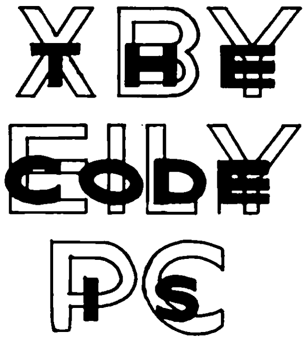
This game uses a randomly selected substitution code. This means that one letter of the alphabet is substituted for another. For instance, the word HELLO may appear as RHUUQ. Here, the R is substituted for the letter H, the H for the letter E, the U for the letter L, and the Q for the letter O. Notice that in the word HELLO, the two Ls are represented by two Us. The substitution holds true throughout the entire mystery phrase. In other words, if U represents the letter L in one word, U will always represent the letter L. A substitution code should not be confused with what is often called a midpoint code. This latter coding method follows the logical progression of the alphabet. However, instead of starting with A, the coded alphabet may start wth S. Given this midpoint coding example, the letter B would then be represented by the letter T, the letter C by the letter U, and so forth. When the end of the midpoint code alphabet is reached at the coded letter Z, the next real letter will be represented by the code letter A and then B, and so forth.

This program uses the RND function of TI BASIC to randomly select a substitute letter for every letter of the standard alphabet. There is no pattern to this selection other than random chance, so midpoint coding is not in effect, and midpoint deciphering techniques will not work. You can, however, gain clues from words that contain apostrophes and also from the repetition of letters in any coded phrase. Also, a single letter that begins a sentence, such as A BOY RAN DOWN THE STREET, can give you a clue to the code. The letter A in this sentence might be coded with the letter T. However, it is used alone, so you might assume that the letter is either an I, as in I RAN DOWN THE STREET, or an A, as in the example above. These are the only two letters in the alphabet that may be used individually in a sentence. From this point on, you should examine the shortest words and then go on to the longest. Some codes may be deciphered in five or ten minutes, while others may take you an hour or more. Each time you run the program, the same messages contained in the data statements will be coded in a different manner.

It is quite easy to remove the data statements and allow a phrase to be input in plain language via the keyboard. This message, which would be represented by R$, would then be put through the standard coding process. This modification would involve two players, one who will input the message to be coded and the other who would tackle the deciphering assignment. On the next turn, the players would switch roles.

Here's how the program works. Lines 100

Listing 29. Crypto.



```
100  REM CRYPTO
110   REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES
120  REM 6/4/83
130  REM PROGRAM RUNS IN TI-B
ASIC
140  CALL CLEAR
150  PRINT "WELCOME TO THE CR
YPTOGRAPHY"
160  PRINT
```

Listing continued.

```
170  PRINT "GAME CALLED 'CRYP
TO'."
180  PRINT
190  PRINT "A SHORT, ENCRYPTE
D PHRASE"
200  PRINT
210  PRINT "WILL APPEAR ON TH
E SCREEN"
220  PRINT
230  PRINT "WHICH YOU MUST DE
CODE. THE"
240  PRINT
250  PRINT "PHRASE IS ENCODED
 USING"
260  PRINT
270  PRINT "A SIMPLE SUBSTITU
TION CODE"
280  PRINT
290  PRINT "SUPPLIED BY THE C
OMPUTER."
300  FOR DLAY=1 TO 1000
310  NEXT DLAY
320  PRINT
330  PRINT "THE CODING METHOD
 IS DIFFER-"
340  PRINT
350  PRINT "ENT EACH TIME.  TY
PE IN YOUR"
360  PRINT
370  PRINT "DECODED PHRASE AN
D THE COM-"
380  PRINT
390  PRINT "PUTER WILL TELL Y
OU IF IT"
400  PRINT
410  PRINT "IS CORRECT.  GOOD
LUCK!"
420  PRINT
430  PRINT
440  PRINT "PRESS (ENTER) TO
CONTINUE."
450  INPUT GH$
460  CALL CLEAR
470  PRINT "          COMPUTI
NG"
480  PRINT
490  PRINT
500  PRINT
510  PRINT
520  PRINT
530  PRINT
540  PRINT
550  PRINT
560  PRINT
570  PRINT
580  RANDOMIZE
590  DIM A(26)
600  DIM B(26)
610  FOR X=1 TO 26
620  A(X)=X+64
630  NEXT X
640  I=INT(RND*26)+1
650  IF A(I)=0 THEN 640
660  Z=Z+1
670  B(Z)=A(I)
680  A(I)=0
690  IF Z=26 THEN 710
700  GOTO 640
710  Z=0
720  CALL CLEAR
730  READ R$
740  W=LEN(R$)
750  FOR C=1 TO W
760  D$=SEG$(R$,C,1)
770  IF D$="." THEN 1070
780  IF D$="?" THEN 1070
```

Listing continued.

```
790   IF D$="'" THEN 1070
800   IF D$=" " THEN 1070
810   IF D$="A" THEN 1090
820   IF D$="B" THEN 1110
830   IF D$="C" THEN 1130
840   IF D$="D" THEN 1150
850   IF D$="E" THEN 1170
860   IF D$="F" THEN 1190
870   IF D$="G" THEN 1210
880   IF D$="H" THEN 1230
890   IF D$="I" THEN 1250
900   IF D$="J" THEN 1270
910   IF D$="K" THEN 1290
920   IF D$="L" THEN 1310
930   IF D$="M" THEN 1330
940   IF D$="N" THEN 1350
950   IF D$="O" THEN 1370
960   IF D$="P" THEN 1390
970   IF D$="Q" THEN 1410
980   IF D$="R" THEN 1430
990   IF D$="S" THEN 1450
1000   IF D$="T" THEN 1470
1010   IF D$="U" THEN 1490
1020   IF D$="V" THEN 1510
1030   IF D$="W" THEN 1530
1040   IF D$="X" THEN 1550
1050   IF D$="Y" THEN 1570
1060   IF D$="Z" THEN 1590
1070   PRINT D$;
1080   GOTO 1600
1090   PRINT CHR$(B(1));
1100   GOTO 1600
1110   PRINT CHR$(B(2));
1120   GOTO 1600
1130   PRINT CHR$(B(3));
1140   GOTO 1600
1150   PRINT CHR$(B(4));
1160   GOTO 1600
1170   PRINT CHR$(B(5));
1180   GOTO 1600
1190   PRINT CHR$(B(6));
1200   GOTO 1600
1210   PRINT CHR$(B(7));
1220   GOTO 1600
1230   PRINT CHR$(B(8));
1240   GOTO 1600
1250   PRINT CHR$(B(9));
1260   GOTO 1600
1270   PRINT CHR$(B(10));
1280   GOTO 1600
1290   PRINT CHR$(B(11));
1300   GOTO 1600
1310   PRINT CHR$(B(12));
1320   GOTO 1600
1330   PRINT CHR$(B(13));
1340   GOTO 1600
1350   PRINT CHR$(B(14));
1360   GOTO 1600
1370   PRINT CHR$(B(15));
1380   GOTO 1600
1390   PRINT CHR$(B(16));
1400   GOTO 1600
1410   PRINT CHR$(B(17));
1420   GOTO 1600
1430   PRINT CHR$(B(18));
1440   GOTO 1600
1450   PRINT CHR$(B(19));
1460   GOTO 1600
1470   PRINT CHR$(B(20));
1480   GOTO 1600
1490   PRINT CHR$(B(21));
1500   GOTO 1600
1510   PRINT CHR$(B(22));
1520   GOTO 1600
1530   PRINT CHR$(B(23));
1540   GOTO 1600
```

Listing continued.

```
1550  PRINT CHRS(B(24));
1560  GOTO 1600
1570  PRINT CHRS(B(25));
1580  GOTO 1600
1590  PRINT CHR$(B(26));
1600  NEXT C
1610  PRINT
1620  PRINT
1630  PRINT
1640  PRINT
1650  PRINT
1660  PRINT "TYPE IN THE DECI
PHERED"
1670  PRINT
1680  PRINT "MESSAGE."
1690  PRINT
1700  PRINT
1710  INPUT QW$
1720  CALL CLEAR
1730  IF QW$=R$ THEN 1830 ELS
E 1740
1740  PRINT "INCORRECT ANSWER
!!"
1750  CALL SOUND(200,-3,0)
1760  PRINT
1770  PRINT
1780  PRINT "THE CORRECT ANSW
ER IS:"
1790  PRINT
1800  PRINT
1810  PRINT R$
1820  GOTO 2030
1830  PRINT R$
1840  PRINT
1850  PRINT "IS A CORRECT ANS
WER!!"
1860  CALL SOUND(250,1500,0)
1870  GOTO 2030
1880  CALL CLEAR
1890  DATA HELLO. HOW ARE YOU
?
1900 DATA THE AIR IS FINE.
1910  DATA SEVEN SONS WALKED
TALL.
1920  DATA IS THIS THE RIGHT
HOUSE?
1930  DATA DON'T YOU KNOW HOW
?
1940  DATA HELEN DID NOT KNOW
.
1950  DATA HE CAME ANYWAY.
1960  DATA WALK. DON'T RUN.
1970  DATA DAISEY'S DOG'S DIN
NER
1980  DATA COLOR OR BLACK AND
 WHITE?
1990  DATA THE RAINS CAME.
2000  DATA NOWHERE BUT IN AME
RICA.
2010  DATA A NOVA IN THE DARK
2020  DATA THE WALLS HAVE EAR
S.
2030  PRINT
2040  PRINT
2050  PRINT "PLAY AGAIN(Y/N)"
2060  PRINT
2070  INPUT AY$
2080  IF AYS="Y" THEN 460
2090  IF AYS="N" THEN 2110
2100  GOTO 2050
2110  CALL CLEAR
2120  PRINT "THANKS FOR PLAYI
NG."
2130  END
```

through 450 initialize the screen and print a short description of how the game is played. Lines 440 and 450 prompt the player to press the enter key to continue and then accept the input. Line 460 clears the screen again, and COMPUTING is printed at the center of the screen. Lines 480 through 570 scroll the word to the exact center. This routine was included because there is a time lag of 15 to 20 seconds while the program arrays are being initialized. This prompt lets the player know that the program is working and to stand by.

The randomize statement is included in line 580 to allow the RND function to return a random pattern of numbers. Without the randomize statement, the same code will be used each time the program is rerun. Two arrays are established in lines 590 and 600. Array A contains the ASCII code numbers for the letters of the alphabet in order from A to Z. Array B will contain the substitution code (again, given in ASCII code numbers), which will be used to encode the data statement messages.

Lines 610 through 630 feed in the sequential alphabet to array A. In line 610, the variable X counts from 1 to 26 (the number of letters in the alphabet). Line 620 assigns each array position the value of X plus 64. In other words, the first array position will be occupied by the number 65 (X + 64), which is the ASCII code for the capital letter A. On the next cycle of the loop, position 2 in array A will be occupied by the number 66, the ASCII code for the capital letter B. When the loop times out, the ASCII codes for the letters A through Z have been fed to array A.

At this point, line 640 assigns a random number from 1 to 26 to variable I. Skip line 650 for the time being. Line 660 is a simple count line that assigns the letter Z to a value equal to itself plus 1. Each time line 660 is executed, Z will step its value by 1. Line 670 begins feeding information to array B. The first position of array B will be filled by the letter contained in position I of array A. If I is equal to 26, the first position of array B will be assigned to the letter Z. In this example, the standard letter A will be substituted by the coded letter Z. Line 680 reassigns position I in array A to the value of zero (0). Skip line 690 for the time being. Line 700 branches to line 640, where another random number is assigned to variable I. Here is where line 650 comes into play. It checks for a value of I that has been previously output by the random number generator. In this case, position A(I) will have been previously assigned a value of zero (O). When this is the case, there is a branch back to line 640 in order to pick up another random number. What line 650 effectively does is tell the computer "This number has been used before, so send me another one." Notice that this branch back takes place before line 660 is executed, so Z is not increased until an unused random number is generated. When Z is equal to 26, the array B has been completely filled with the substitution code. Line 690 detects the value of 26 for the variable Z and branches to line 710. Here, Z is reassigned the value of zero (0) in order to set up for the next substitution code, which will be established after the player has input his or her guess. Line 720 clears the screen again.

Line 730 reads the first data statement line, which contains the message to be encoded. Line 740 assigns the number of characters and spaces contained in the data statement line to the variable W. In line 750 a for-next loop that counts from 1 to W is entered. Line 760 assigns the first character in the data line to the string variable D$ and then steps by one character at a time each time the loop is cycled. The SEG$ function is used to read each letter, space, and punctuation mark contained in the data statement line on an individual basis. Remember, the loop will count from 1 to the required number of cycles to read the entire data message. Lines 770 through 1060 perform the encoding function. If D$ is equal to a period (.), question mark (?), apostrophe ('), or space ( ), lines 770 through 800 will detect this condition and will branch to line 1070. This line simply prints D$ on the screen. In other words, periods, question marks, apostrophes, and spaces contained in the data statement line will not be encoded. The letters of the alphabet, however, are handled in a completely different manner.

Let's assume that the first letter in the data statement phrase is an A. Line 810 detects this and branches to line 1090. Here, there is an instruction to print the letter equivalent of the ASCII code found in array B at position 1: PRINT CHR$(B(1));. Whatever letter has been assigned to position 1 of array B will always represent the letter A. There is then a branch to line 1600, which causes the loop to recycle.

Any other letters will be treated in the same manner, except that there will be branches to different lines for each different letter. When this loop times out, all of the characters contained in the original data statement line will have been printed on the screen in encoded form. Lines 1610 through 1650 cause the coded message to be scrolled toward the center of the screen, while line 1660 begins printing the prompt telling the player to type in his deciphered message.

The deciphered message is input at line 1710 and is assigned to the variable QW$. Line 1720 then clears the screen, while line 1730 compares the player's input with the original data statement line represented by R$. If the two are equal, there is a branch to line 1830, which prints R$ on the screen. Subsequent lines print the fact that this is the correct answer. Line 1860 uses the Call Sound subprogram to produce a pleasing note output from the monitor speaker. There is then a branch to line 2030, which eventually brings about the PLAY AGAIN prompt, allowing you to have another message displayed using a completely different code.

In line 1730, if QW$ is not equal to R$, there is a branch to line 1740, which displays the fact that your answer is incorrect, and an unpleasant sound is produced in line 1750. Line 1780 prints THE CORRECT ANSWER IS:. Line 1810 then prints R$, which is the correct answer. Again, there is a branch to line 2030, which asks if you want to play again. Line 2070 receives the player input at this point, which should be either Y or N. If a Y is input, indicating that you do wish to play again, line 2080 branches to line 460, where the screen is cleared and the two arrays are once again initialized. This time, however, array B will contain a different coding structure because of the use of the RND function. If the player does not wish to play again, he or she inputs the letter N. This causes line 2090 to branch to line 2110, where the exit message is displayed. If neither Y or N is input, line 2100 simply branches back to the PLAY AGAIN prompt.

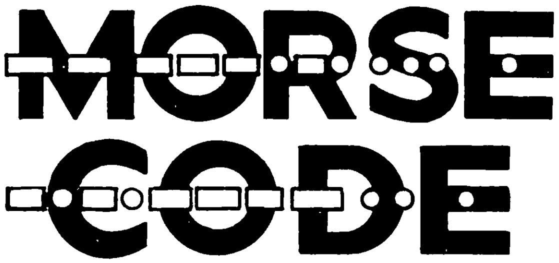
This is an efficient program and offers an almost limitless number of encoding possibilities. Most programmers will want to change or modify the fourteen data lines included in lines 1890 through 2020. You can certainly add as many data lines as you want, and you can change the message in any manner. Do not use any punctuation marks other than the period, question mark, or apostrophe. These will simply delay the time it takes to display the encoded message. If you wish to use exclamation points, quotation marks, etc., these can be included by simply adding more lines between the present lines 770 and 790, which assign D$ to these characters and then branch to line 1070 to print D$ as it is.

This program does not contain a routine to detect the end of the last data message. Therefore, after you have gone through the fourteen data statements provided, an error message will occur when you attempt the fifteenth run. If you wish, you can add one more data statement line at the end of those already provided. This line should contain the word END. Then, add a line 735 that reads IF R$="END" THEN 2110. When the word END is detected, the branch to line 2110 will clear the screen, and the exit message will appear.

MORSE PUZZLER

Morse Puzzler is an educational game that can help you learn Morse Code or improve your skills in this communications medium. The program uses several plain text data statement lines and outputs the letter information in Morse Code using the Call Sound subprogram in TI BASIC. The dots and dashes are heard from your monitor's speaker. The data statement lines begin at program line 3650 and can be altered to include any type of message you desire.

Listing 30. Morse Code



```
100  REM MORSE PUZZLER
110  REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES
120  REM 6/6/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  PRINT "THIS IS A MORSE C
ODE SKILL"
160  PRINT
170  PRINT "GAME. IT WILL HEL
P YOU TO"
180  PRINT
190  PRINT "MASTER MORSE CODE
 AND COM-"
200  PRINT
210  PRINT "PETE AGAINST YOUR
SELF OR"
220  PRINT
```

Listing continued.

```
230  PRINT "ANOTHER HUMAN PLA
YER AT"
240  PRINT
250  PRINT "THE SAME TIME. TH
E COMPUTER"
260  PRINT
270  PRINT "WILL TRANSMIT A S
ERIES IN"
280  PRINT
290  PRINT "MORSE CODE. DECOD
E THE MES-"
300  PRINT
310  PRINT "SAGE AND THEN TYP
E IT IN."
320  PRINT
330  PRINT "THE COMPUTER WILL
 TELL YOU"
340  PRINT
350  PRINT "IF YOUR DECODING
IS CORRECT."
360  PRINT
370  PRINT "PRESS (ENTER) TO
BEGIN."
380  INPUT EW$
390  CALL CLEAR
400  READ A$
410  IF A$="END" THEN 3850
420  L=LEN(A$)
430  FOR X=1 TO L
440  C$=SEG$(A$,X,1)
450  IF C$="A" THEN 720
460  IF C$="B" THEN 790
470  IF C$="C" THEN 920
480  IF C$="D" THEN 1050
490  IF C$="E" THEN 1150
500  IF C$="F" THEN 1190
510  IF C$="G" THEN 1320
520  IF C$="H" THEN 1420
530  IF C$="I" THEN 1550
540  IF C$="J" THEN 1620
550  IF C$="K" THEN 1750
560  IF C$="L" THEN 1850
570  IF C$="M" THEN 1980
580  IF C$="N" THEN 2050
590  IF C$="O" THEN 2120
600  IF C$="P" THEN 2220
610  IF C$="Q" THEN 2350
620  IF C$="R" THEN 2480
630  IF C$="S" THEN 2580
640  IF C$="T" THEN 2680
650  IF C$="U" THEN 2720
660  IF C$="V" THEN 2820
670  IF C$="W" THEN 2950
680  IF C$="X" THEN 3050
690  IF C$="Y" THEN 3180
700  IF C$="Z" THEN 3310
710  IF C$=" " THEN 3430 ELSE
 3450
720  CALL SOUND(150,1500,3)
730  FOR XX=1 TO 50
740  NEXT XX
750  CALL SOUND(350,1500,3)
760  FOR XX=1 TO 200
770  NEXT XX
780  GOTO 3450
790  CALL SOUND(350,1500,3)
800  FOR XX=1 TO 100
810  NEXT XX
820  CALL SOUND(150,1500,3)
830  FOR XX=1 TO 50
840  NEXT XX
850  CALL SOUND(150,1500,3)
860  FOR XX=1 TO 50
870  NEXT XX
880  CALL SOUND(150,1500,3)
890  FOR XX=1 TO 200
```

Listing continued.

```
900   NEXT XX
910   GOTO 3450
920   CALL SOUND(350,1500,3)
930   FOR XX=1 TO 100
940   NEXT XX
950   CALL SOUND(150,1500,3)
960   FOR XX=1 TO 50
970   NEXT XX
980   CALL SOUND(350,1500,3)
990   FOR XX=1 TO 100
1000  NEXT XX
1010  CALL SOUND(150,1500,3)
1020  FOR XX=1 TO 200
1030  NEXT XX
1040  GOTO 3450
1050  CALL SOUND(350,1500,3)
1060  FOR XX=1 TO 100
1070  NEXT XX
1080  CALL SOUND(150,1500,3)
1090  FOR XX=1 TO 50
1100  NEXT XX
1110  CALL SOUND(150,1500,3)
1120  FOR XX=1 TO 200
1130  NEXT XX
1140  GOTO 3450
1150  CALL SOUND(100,1500,3)
1160  FOR XX=1 to 200
1170  NEXT XX
1180  GOTO 3450
1190  CALL SOUND(150,1500,3)
1200  FOR XX=1 TO 50
1210  NEXT XX
1220  CALL SOUND(150,1500,3)
1230  FOR XX=1 TO 50
1240  NEXT XX
1250  CALL SOUND(350,1500,3)
1260  FOR XX=1 TO 100
1270  NEXT XX
1280  CALL SOUND(150,1500,3)
1290  FOR XX=1 TO 200
1300  NEXT XX
1310  GOTO 3450
1320  CALL SOUND(350,1500,3)
1330  FOR XX=1 TO 100
1340  NEXT XX
1350  CALL SOUND(350,1500,3)
1360  FOR XX=1 TO 100
1370  NEXT XX
1380  CALL SOUND(150,1500,3)
1390  FOR XX=1 TO 200
1400  NEXT XX
1410  GOTO 3450
1420  CALL SOUND(150,1500,3)
1430  FOR XX=1 TO 50
1440  NEXT XX
1450  CALL SOUND(150,1500,3)
1460  FOR XX=1 TO 50
1470  NEXT XX
1480  CALL SOUND(150,1500,3)
1490  FOR XX=1 TO 50
1500  NEXT XX
1510  CALL SOUND(150,1500,3)
1520  FOR XX=1 TO 200
1530  NEXT XX
1540  GOTO 3450
1550  CALL SOUND(150,1500,3)
1560  FOR XX=1 TO 50
1570  NEXT XX
1580  CALL SOUND(150,1500,3)
1590  FOR XX=1 TO 200
1600  NEXT XX
1610  GOTO 3450
1620  CALL SOUND(150,1500,3)
1630  FOR XX=1 TO 50
1640  NEXT XX
1650  CALL SOUND(350,1500,3)
```

Listing continued.

```
1660  FOR XX=1 TO 100
1670  NEXT XX
1680  CALL SOUND(350,1500,3)
1690  FOR XX=1 TO 100
1700  NEXT XX
1710  CALL SOUND(350,1500,3)
1720  FOR XX=1 TO 200
1730  NEXT XX
1740  GOTO 3450
1750  CALL SOUND(350,1500,3)
1760  FOR XX=1 TO 100
1770  NEXT XX
1780  CALL SOUND(150,1500,3)
1790  FOR XX=1 TO 50
1800  NEXT XX
1810  CALL SOUND(350,1500,3)
1820  FOR XX=1 TO 200
1830  NEXT XX
1840  GOTO 3450
1850  CALL SOUND(150,1500,3)
1860  FOR XX=1 TO 50
1870  NEXT XX
1880  CALL SOUND(350,1500,3)
1890  FOR XX=1 TO 100
1900  NEXT XX
1910  CALL SOUND(150,1500,3)
1920  FOR XX=1 TO 50
1930  NEXT XX
1940  CALL SOUND(150,1500,3)
1950  FOR XX=1 TO 200
1960  NEXT XX
1970  GOTO 3450
1980  CALL SOUND(350,1500,3)
1990  FOR XX=1 TO 100
2000  NEXT XX
2010  CALL SOUND(350,1500,3)
2020  FOR XX=1 TO 200
2030  NEXT XX
2040  GOTO 3450
2050  CALL SOUND(350,1500,3)
2060  FOR XX=1 TO 100
2070  NEXT XX
2080  CALL SOUND(150,1500,3)
2090  FOR XX=1 TO 200
2100  NEXT XX
2110  GOTO 3450
2120  CALL SOUND(350,1500,3)
2130  FOR XX=1 TO 100
2140  NEXT XX
2150  CALL SOUND(350,1500,3)
2160  FOR XX=1 TO 100
2170  NEXT XX
2180  CALL SOUND(350,1500,3)
2190  FOR XX=1 TO 200
2200  NEXT XX
2210  GOTO 3450
2220  CALL SOUND(150,1500,3)
2230  FOR XX=1 TO 50
2240  NEXT XX
2250  CALL SOUND(350,1500,3)
2260  FOR XX=1 TO 100
2270  NEXT XX
2280  CALL SOUND(350,1500,3)
2290  FOR XX=1 TO 100
2300  NEXT XX
2310  CALL SOUND(150,1500,3)
2320  FOR XX=1 TO 200
2330  NEXT XX
2340  GOTO 3450
2350  CALL SOUND(350,1500,3)
2360  FOR XX=1 TO 100
2370  NEXT XX
2380  CALL SOUND(350,1500,3)
2390  FOR XX=1 TO 100
2400  NEXT XX
2410  CALL SOUND(150,1500,3)
```

Listing continued.

```
2420  FOR XX=1 TO 50
2430  NEXT XX
2440  CALL SOUND(350,1500,3)
2450  FOR XX=1 TO 200
2460  NEXT XX
2470  GOTO 3450
2480  CALL SOUND(150,1500,3)
2490  FOR XX=1 TO 50
2500  NEXT XX
2510  CALL SOUND(350,1500,3)
2520  FOR XX=1 TO 100
2530  NEXT XX
2540  CALL SOUND(150,1500,3)
2550  FOR XX=1 TO 200
2560  NEXT XX
2570  GOTO 3450
2580  CALL SOUND(150,1500,3)
2590  FOR XX=1 TO 50
2600  NEXT XX
2610  CALL SOUND(150,1500,3)
2620  FOR XX=1 TO 50
2630  NEXT XX
2640  CALL SOUND(150,1500,3)
2650  FOR XX=1 TO 200
2660  NEXT XX
2670  GOTO 3450
2680  CALL SOUND(350,1500,3)
2690  FOR XX=1 TO 200
2700  NEXT XX
2710  GOTO 3450
2720  CALL SOUND(150,1500,3)
2730  FOR XX=1 TO 50
2740  NEXT XX
2750  CALL SOUND(150,1500,3)
2760  FOR XX=1 TO 50
2770  NEXT XX
2780  CALL SOUND(350,1500,3)
2790  FOR XX=1 TO 200
2800  NEXT XX
2810  GOTO 3450
2820  CALL SOUND(150,1500,3)
2830  FOR XX=1 TO 50
2840  NEXT XX
2850  CALL SOUND(150,1500,3)
2860  FOR XX=1 TO 50
2870  NEXT XX
2880  CALL SOUND(150,1500,3)
2890  FOR XX=1 TO 50
2900  NEXT XX
2910  CALL SOUND(350,1500,3)
2920  FOR XX=1 TO 200
2930  NEXT XX
2940  GOTO 3450
2950  CALL SOUND(150,1500,3)
2960  FOR XX=1 TO 50
2970  NEXT XX
2980  CALL SOUND(350,1500,3)
2990  FOR XX=1 TO 100
3000  NEXT XX
3010  CALL SOUND(350,1500,3)
3020  FOR XX=1 TO 200
3030  NEXT XX
3040  GOTO 3450
3050  CALL SOUND(350,1500,3)
3060  FOR XX=1 TO 100
3070  NEXT XX
3080  CALL SOUND(150,1500,3)
3090  FOR XX=1 TO 50
3100  NEXT XX
3110  CALL SOUND(150,1500,3)
3120  FOR XX=1 TO 50
3130  NEXT XX
3140  CALL SOUND(350,1500,3)
3150  FOR XX=1 TO 200
3160  NEXT XX
3170  GOTO 3450
```

Listing continued.

```
3180  CALL SOUND(350,1500,3)
3190  FOR XX=1 TO 100
3200  NEXT XX
3210  CALL SOUND(150,1500,3)
3220  FOR XX=1 TO 50
3230  NEXT XX
3240  CALL SOUND(350,1500,3)
3250  FOR XX=1 TO 100
3260  NEXT XX
3270  CALL SOUND(350,1500,3)
3280  FOR XX=1 TO 200
3290  NEXT XX
3300  GOTO 3450
3310  CALL SOUND(350,1500,3)
3320  FOR XX=1 TO 100
3330  NEXT XX
3340  CALL SOUND(350,1500,3)
3350  FOR XX=1 TO 100
3360  NEXT XX
3370  CALL SOUND(150,1500,3)
3380  FOR XX=1 TO 50
3390  NEXT XX
3400  CALL SOUND(150,1500,3)
3410  FOR XX=1 TO 200
3420  NEXT XX
3430  FOR XX=1 TO 100
3440  NEXT XX
3450  NEXT X
3460  PRINT "PRINT THE DECODE
D MESSAGE"
3470  INPUT QW$
3480  CALL CLEAR
3490  IF QW$=A$ THEN 3600
3500  PRINT "THE CORRECT ANSW
ER IS:"
3510  PRINT
3520  PRINT
3530  PRINT A$
3540  PRINT
3550  PRINT
3560  PRINT
3570  PRINT "PRESS (ENTER) TO
 CONTINUE"
3580  INPUT EW$
3590  GOTO 3640
3600  PRINT "THAT ANSWER IS C
ORRECT!!"
3610  PRINT
3620  PRINT "PRESS (ENTER) TO
 CONTINUE"
3630  INPUT EW$
3640  GOTO 390
3650  DATA HOW IS THE WEATHER
 IN SPAIN
3660  DATA THE QUICK BROWN FO
X JUMPED OVER THE LAZY DOG
3670  DATA THE GRAY RABBIT WA
S NEVER CAUGHT
3680  DATA ELECTRON TUBES HAV
E BEEN REPLACED BY TRANSISTO
RS
3690  DATA ELEPHANTS ARE BIGG
ER THAN SATELLITES
3700  DATA RHOMBIC ANTENNAS A
RE SUPERIOR
3710  DATA MARTIANS ARE ALIEN
 TO VENERIANS
3720  DATA THE HYPERBOLA CONS
ISTS OF TWO CURVES
3730  DATA MINIATURE CIRCUITS
 ARE ALSO KNOWN AS MICRO CIR
CUITS
3740  DATA FREE FALLING IS A
CHALLENGING SPORT
3750  DATA THE WIRELESS WAS I
NVENTED BY MARCONI
```

Listing continued.

```
3760  DATA CALLING CQ RESULTS
 IN A CHANCE COMMUNICATION
3770  DATA FAR BE IT FROM THE
M TO TELL
3780  DATA RUNNING IN THE DEL
TA WAS THE WILY HARE
3790  DATA AMATEUR RADIO INVO
LVES SEVERAL LICENSE CLASSES
3800  DATA WATER OVER THE DAM
 MEANS IT IS TOO LATE TO WORRY
3810  DATA WHEN THE RAINS COM
E THE GROUNDS ARE POROUS
3820  DATA WHAT IS THE ANSWER
 TO THIS RIDDLE
3830  DATA TO BE OR NOT TO BE
 IS THE QUESTION
3840  DATA ROLLING STONES GAT
HER NO MORSE,END
3850  END
```

Lines 100-380 clear the screen and give a brief explanation as to how the game works. When you press the enter key, line 390 clears the screen again, while line 400 reads the first data statement line. The data statement information is assigned to the string variable A$. Line 410 tests for the final data statement line. If the information contained here is the word END, there is a branch to line 3850 and the end statement.

Line 420 reads the number of characters in the data statement line. Line 430 establishes the major loop in the program, which counts from 1 to L, the latter variable having been assigned the number of characters in A$. Line 440 uses the SEG$ function to read each character in A$ individually. Each character is assigned to the variable C$. Lines 450 through 710 test for the letter value of C$, making appropriate branches to the actual Morse Coding lines, which begin at program line 720.

Lines 720 through 3440 contain all of the information needed to generate each letter of the alphabet in Morse Code. I chose a frequency of 1500 hertz. This figure is contained in the Call Sound subprogram lines. By experimentation, I decided to use 150 as the duration for a dot and 350 as the duration for the dash.

Let's take a practical example. Assume that A$ is equal to A DOG RAN DOWN THE STREET. This entire phrase is 25 characters in length including the spaces between words. Line 420 assigns the value 25 to L, so the loop in line 430 will count from 1 to 25 in order to read each letter in A$. During the first pass of the loop, line 440 will assign C$ the value of A, since this is the first letter in the A$ phrase. Line 450 branches to line 720, since C$ does equal "A". Lines 720 through 770 contain the coding information for the letter A, which is represented in Morse Code by a dot and then a dash. In line 720, the Call Sound subprogram is set up to produce a 1500 hertz tone for a duration of 150, which is a fraction of a second. In Morse Code, however, the pauses between the dots and dashes are significant. Therefore, lines 730 and 740 contain a delay sequence while the computer counts from 1 to 50. This count is used to separate dots from other dots or dots from dashes. A delay count of 100 is used to separate dashes from dots or dashes from other dashes. Each letter is followed by a count of 200 in order to separate it from the next letter.

Line 750 contains the Call Sound Subprogram for a dash, which completes the letter A. Since this is the last part of the letter sequence, lines 760 and 770 contain the sequence end delay, a count from 1 to 200. Line 780 branches to line 3450, which contains a next statement that recycles the X loop. This effectively branches back to line 430, where the loop cycles again with X being equal to 2. Line 440 samples the second character in A$, which in this case is a space between the first word in the sentence, A, and the next word, which begins with the letter D. C$ is therefore equal to a space. This condition is detected in line 710, which branches to line 3430. A space is represented by another time delay. Here, XX counts from 1 to 100. This gives a longer delay than the space between letters, even though the delay between letters counts from 1 to 200 because the if-then line that tests for a space lies at the end of all of the if-then lines for the letters of the alphabet. An automatic delay is thus built in.

Once this delay loop times out, the X loop again recycles, picking up the third character of A$, which is the letter D. Line 480 detects the letter D and branches to line 1050, where the Call Sound subprograms are used with the delay loops to produce the code sequence dash-dot-dot, the Morse Code for the letter D. Line 1050 contains the Call Sound subprogram for a dash, while lines 1060 and 1070 contain the dash delay, a count of from 1 to 100. Line 1080 contains the Call Sound subprogram for a dot. Lines 1090 and 1100 contain the dot separation delay, a count of from 1 to 50. Lines 110 through 1130 are identical to the previous three because another dot must be formed. However, you will notice that the count in line 1120 is from 1 to 200, since this last dot completes the letter.

This same sequence of events will take place for all the characters in A$ until the last one is sounded. When this is done, the X loop times out and line 3460 displays a prompt on the screen telling you to type in the decoded message. Your input is committed to the string variable QW$ in line 3470 and line 3840 clears the screen. Line 3490 tests to see if you input (QW$) is equal to the original message (A$). If this is so, there is a branch to line 3600, which tells you the answer is correct and instructs you to press the enter key to continue. When you press the enter key, there is another branch to line 390, which again clears the screen. At this point, line 400 reads the next data statement line and a new phrase is output in Morse Code.

If QW$ in line 3490 is not equal to A$, line 3500 is executed, which eventually brings about the printing of the correct answer on the screen. I have included several data statement message lines to get the program going, but of course, you can include any text in these lines that you desire. This program does not address the coding of numbers and the various punctuation marks that are allowed in the Morse Code set. This program is designed to be fun and educational and to introduce the player to Morse Code. With few additional lines, numbers and punctuation marks can be included, however.

Some persons will undoubtedly wish to

forego the data statement lines in order to give themselves the opportunity to input a line directly via the keyboard while the program is running. This can be easily accomplished by simply replacing line 400 with:

```
400  INPUT A$
```

When this is done, execution will be temporarily halted while you type in a phrase. When you press the enter key, whatever you typed in will be run through the coding sequence.

This is quite a long program to write. It could have been shortened considerably by using GOSUB branches to single lines containing the information for the dot, the spacing after the dot, the dash, the spacing after the dash, and finally, the end of character spacing routine. Such a program is more difficult to understand, however, than this one, which uses straight-line programming. Once this program has been input as shown, you can input any phrase information you desire and have it come out in Morse Code.

As written, this program sends code at the rate of about eight words per minute. You can speed things up (or slow them down) by changing the duration of the delay between the dots and dashes, and possibly changing the duration of the dot and dash proportionately. Increase the count value of your for-next loops used for delay purposes to slow sending speed. Decrease these values to increase it.

All in all, I think you will find the program to be most interesting and highly enjoyable if you have a desire to learn Morse Code or are already somewhat familiar with this means of communication. Undoubtedly, some amateur radio operators will use a program of this type to send Morse Code via their transmitters while typing the information in a letter form via the TI-99/4A keyboard. This program has many applications outside of the game programming environment.

TIME KEEPING PROGRAMS

In some instances, it's desirable to have some means of keeping time in game programs. For example, any of the word puzzle programs presented might be more interesting if each player were limited to a certain number of seconds or minutes to find the words. Adding a countdown feature to one of these programs simply involves putting the game itself on the screen and then displaying numbers that count backward from a figure you choose. When 0 is reached, the computer would announce that time is up, and it would be another player's turn.

Many personal computers contain TIME$ functions, which use the internal clock of the computer in a machine language program that will automatically keep time. This program is run internally. Therefore, once the correct time has been input, you may run any programs you wish and return to the TIME$ function at any time and receive the correct readout.

Like most home computers, the TI-99/4A does not contain a TIME$ function. There is no practical way to access the internal clock to simulate this function, but it can be simulated during the run of a program designed specifically to tell time. Again, the programs I'm speaking of can be set up to accurately display time, but when the program is terminated, the timekeeping function is immediately discontinued.

Anyone who has programmed even for a short period of time knows that when you crowd a lot of statements into a for-next loop, a time lag occurs. This can be seen in the flicker of moving graphic objects on the screen, for example. Many of the programs presented in this chapter have used time delay loops which simply count from 1 to a high number and then move on to another routine. Since it is easy to generate time delays, one could also assume that it's fairly easy to generate these delays in units which correspond to seconds. This is a correct assumption.

The following programs are not games in themselves, but they can be used to spice up some game programs, as well as for many other purposes. By adding a few program lines you can even use some of them to teach youngsters to tell time.

Ten Second Timer

This program is designed to count from 0 to 10 in a period of 10 seconds. The numerals will be displayed on the screen.

In this simple program line 140 clears the screen, and line 150 begins a for-next loop that counts from 49 to 57. These two numbers represent the ASCII codes for the numerals 1 and 9. The numbers in between make up the numerals 2 through 8. Line 160 displays these numerals on the screen by feeding the ASCII equivalent into a Call HCHAR subprogram. The heart of the clock program, however, is found in lines 170 and 180. Through experimentation, I determined that it took the TI-99/4A one second to count from 1 to 325 while running in TI BASIC. If you are running TI Advanced BASIC, execution time will be different and you will have to adjust the count in line 170 with the aid of an accurate watch. Lines 170 and 180 form a time delay loop. When this times out, line 190 is executed and the loop begun in line 150 recycles.

However, the top of the loop, or the number 57, is the ASCII code for the number 9. This program is to count to a maximum of 10. Lines 200 and 210 are executed after the loop in lines 170 and 180 and the master loop in lines 150 and 190 time out. Lines 200 and 210 display the number 10 on the screen and the program is terminated.

Again, the for-next loop in lines 170 and

Listing 31. Ten Second Timer

```
100  REM TEN SECOND TIMER
110  REM COPYRIGHT FREDERICK
 HOLTZ  AND ASSOCIATES
120  REM 4/30/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  FOR X=49 TO 57
160  CALL HCHAR(10,16,X)
170  FOR Y=1 TO 325
180  NEXT Y
190  NEXT X
200  CALL HCHAR(10,15,49)
210  CALL HCHAR(10,16,48)
```

180 determines the accuracy of your clock program. You may not find my values here to give you the accuracy you desire on *your* TI-99/4A, since internal differences (and even temperature) can affect the actual sequence time. As soon as you set up your program, check its accuracy with a stopwatch. For game purposes, if you're off a half second or so, it's not that important. But naturally, you will want the program to be as accurate as possible.

Audible Ten Second Timer

This program is almost identical to the previous one, and you will see the familiar timing loop at lines 160 and 170. In this case, however, the master loop, which is begun at line 150, counts from 1 to 10 (cycles 10 times). During each cycle, the Call Sound subprogram in line 180 will produce a beep. This loop will cycle 10 times, and after each second, another 1000-Hz beep will be heard. The Call Sound subprogram can also be directly inserted into the previous program (line 161) to provide a combination of visual and audible time display. The addition of the Call Sound subprogram will slow execution marginally, however, so it will probably be necessary to decrease the top count of the for-next loop in the previous program.

Listing 32. Audible Ten Second Timer

```
100  REM AUDIBLE TEN SECOND TIMER
110  REM COPYRIGHT FREDERICK
  HOLTZ AND ASSOCIATES
120  REM 4/30/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  FOR A=1 TO 10
160  FOR X=1 TO 325
170  NEXT X
180  CALL SOUND(10,1000,0)
190  NEXT A
```

Countdown Timer

Here's a program that is useful for playing various games. It is a countdown timer that will count from a maximum of 99 seconds down to 0 seconds. You may input any value you want, up to 99 seconds, and get an accurate countdown on the display screen.

Listing 33. Countdown Timer

```
100  REM COUNTDOWN TIMER
110  REM COPYRIGHT FREDERICK
  HOLTZ AND ASSOCIATES
120  REM 5/1/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  INPUT "SECONDS=":A$
```

Listing continued.

```
160  CALL CLEAR
170  A=LEN(A$)
180  IF A> 2 THEN 190 ELSE 240
190  CALL CLEAR
200  PRINT "MAXIMUM TIME IS 9
9 SECONDS"
210  FOR TD=1 TO 1000
220  NEXT TD
230  GOTO 140
240  FOR X=1 TO A
250  AA=ASC(SEG$(A$,X,1))
260  CALL HCHAR(10,14+X,AA)
270  NEXT X
280  CALL GCHAR(10,15,FF)
290  CALL GCHAR(10,16,GG)
300  GG=GG-1
310  FOR DLAY=1 TO 225
320  NEXT DLAY
330  IF GG  48 THEN 340 ELSE 3
70
340  IF FF=48 THEN 400
350  GG=57
360  FF=FF-1
370  CALL HCHAR(10,15,FF)
380  CALL HCHAR(10,16,GG)
390  GOTO 280
400  CALL SOUND(1000,1000,0)
410  END
```

The for-next loop that serves as the timer element in this program is found in lines 310 and 320. The master loop begins at line 280 and recycles at line 390. You can see that there are a large number of statement lines within this loop, and therefore, the delay (DLAY) count has been reduced to 225. This reflects the slowed execution time caused by the extra statement lines. Line 340 branches to line 400 when the clock has counted down to 0. At this time, an alarm which is provided by the Call Sound subprogram in line 400 goes off. This line generates a 1000-Hz tone at a very respectable volume, indicating that a timing sequence has been completed.

Clock

It is only natural to go one step further and program a complete 24-hour clock. Again, this is not a game program in itself, but it can be used as a teaching game when it is necessary for small children to learn to tell time by means of an electronic clock.

When the program is run, you will be asked to input the hours and the minutes. This is a 24-hour clock, so 6:00 A.M. is input as 0.6, while 6:00 P.M. is input as 1, 8. The latter stands for 1800 hours, which is 6:00 P.M. in 24-hour time. The commas are necessary between each numeral to the INPUT statements in lines 150 and 160 which require the inputting of two separate values.

You should always set this clock at the beginning of a new minute. In other words, input the hours numerals and then input the minutes numerals for the upcoming minute. As soon as your watch reads one second before the beginning of a new minute, press the enter key and your clock is activated. You will then see a display such as

```
09:23:14
```

This indicates that the time is twenty-three minutes and 14 seconds past the hour of 9 in the morning. Every second the right-hand number will advance by 1. When it reaches 60, the center number will be stepped by 1. As long as your computer is left on, this program will continue to tell time for you.

The timing loop for this program is found in lines 350 and 360. You will notice that the top value is even lower here than in the preceding programs because of the extra statement lines within the loop. This program combines almost everything found in the first one that was discussed, but it also makes provision for graphically displaying minutes and hours as well as the seconds.

The clock programs presented will be accurate only when run in standard TI BASIC.

Listing 34. Clock

```
100  REM CLOCK
110  REM COPYRIGHT FREDERICK
 HOLTZ AND ASSOCIATES
120  REM 4/28/83
130  REM PROGRAM RUNS IN TI-
BASIC
140  CALL CLEAR
150  INPUT "HOURS":A$,B$
160  INPUT "MINUTES":C$,D$
170  AA=ASC(A$)
180  BB=ASC(B$)
190  CC=ASC(C$)
200  DD=ASC(D$)
210  A=AA-48
220  B=BB-48
230  C=CC-48
240  D=DD-48
250  CALL CLEAR
260  FOR X=48 TO 57
270  CALL HCHAR(12,13,48+A)
280  CALL HCHAR(12,14,48+B)
290  CALL HCHAR(12,15,58)
300  CALL HCHAR(12,16,48+C)
310  CALL HCHAR(12,17,48+D)
320  CALL HCHAR(12,18,58)
330  CALL HCHAR(12,20,X)
340  CALL HCHAR(12,19,48+E)
350  FOR Y=1 TO 180
360  NEXT Y
370  NEXT X
380  E=E+1
390  IF E<>6 THEN 580
400  E=0
410  D=D+1
420  IF D=10 THEN 430 ELSE 58
0
430  D=0
440  C=C+1
450  IF C=6 THEN 460 ELSE 580
460  C=0
470  B=B+1
480  IF A=2 THEN 490 ELSE 500
490  IF B=4 THEN 530 ELSE 500
500  IF B=10 THEN 510 ELSE 58
510  B=0
520  GOTO 580
530  A=0
540  B=0
550  C=0
560  D=0
570  E=0
580  GOTO 260
```

Again, if you elect to use Extended BASIC, change the count values of the timing for-next loops. Even in standard TI BASIC, it may be necessary to add or subtract from the top count of the timing loops in order to make up for internal differences and temperature variations, which can affect execution time.

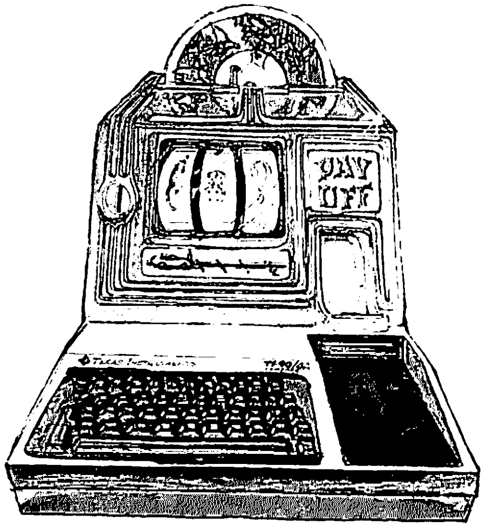
THE PURPOSE OF THE PROGRAMS

The programs presented in this chapter have been designed with several criteria in mind. First, they must be interesting and varied. Second, they must run on a basic TI-99/4A computer without any options. This means that when you purchase your TI-99/4A, you can be assured that all you need do is input these programs, debug any input errors, and then enjoy a successful run. The third criteria involves the instructive properties of each program. Each program must be able to teach the beginner a little more about how his or her machine responds to the BASIC language. The fourth criteria is as important as the previous three. Each program must not be so long as to require many hours of input time. When fatigue sets in, typing errors are made and sometimes, this results in a program that is just about impossible to debug.

I hope you will agree that the programs in this chapter have met all of the above criteria in most instances. Another important criterium involves the modification and/or expansion properties of each program. None of the programs presented here is too complicated for the programmer who has been at it for three months or more. However, these experienced programmers may be able to utilize the programming concepts discussed here and expand upon each of the programs, turning something simple and enjoyable into something complex and more enjoyable.

As you can tell by analyzing these programs, the TI-99/4A home computer is undoubtedly one of the best buys in the computer world today when you consider its extremely low price and wide range of capabilities. These computers will most certainly introduce many thousands of persons to the age of logic in a most successful manner.

Chapter 5



Commercial Game Software for the TI-99/4A

The TI-99/4A computer was first introduced as the TI-99/4 in 1979. Since the machine has been out a few years now, there is a proliferation of software available. Software for this machine is available on three different storage mediums. These include command module, cassette tape, and disk. Some programs may be available on two types of media; while others may be available on only one of the three. The command modules plug into the module slot in the console unit. Cassette tapes are interfaced with the console unit by an optional cassette cable. Software which comes on 5¼″ disks can only be loaded into the machine when it is equipped with the peripheral expansion system, disk adapter, and disk drive.

While the commercial software available for the TI-99/4A includes programs that address home and business needs, you will also find a large variety of entertainment and game programs. This chapter will describe some of them. You may be interested in purchasing some of this software and then modifying the game ideas they provide to develop your own programs. I'm not saying that you should copy the game ideas; only that you should let the multitude of games commercially available for this machine serve to stimulate your own creative juices. Perhaps you can come up with a game that combines the best of all the many currently available games while adding a new flair or bit of excitement.

CARD GAMES

Draw Poker pits one person against the computer. The computer is the dealer. You have a number of options once the cards are dealt, such as raise, call, fold, discard, etc. You are able to see all your cards, while the computer's cards are all face down. You are given a bankroll at the beginning that is equal to the computer's bankroll. The game is over when you or the computer run out of money. This game requires Extended BASIC and is available on cassette (PHT 6037) for $25.00 or disk (PHD 5037) for approximately $30.00.

Challenge Poker, designed for use by up to four players, can be played against the computer as well. Points are scored by creating the best poker hand. The winner is the player who is able to amass 100 points first. Designed by Pewterware, the game is available on cassette only (CPW67070).

Casino Pack was developed by Ehninger Associates, Inc. In this game, the computer is the house and you are betting. The package includes a slot machine and blackjack tables. Based on the popular Las Vegas game, this package is available on disk only (D1010).

Blackjack was developed by Color Software, and up to four may play at the same time. All the cards in a standard deck are displayed face up in a colorful and easy-to-recognize manner. The game is designed for users 12 and older and is available on cassette only.

Developed by Milton Bradley Company, the **Blackjack and Poker** package contains two betting games with which most people are familiar. You are given a certain amount of money at the beginning of each game. When your money runs out, the game is over. Both games can be played by users aged 10 and older, and the package is available as a plug-in module for about $25.00 (PHM 3033).

SPORTS GAMES

All* Star Baseball, designed by Ehninger Associates, Inc., is a two-player game in which each player has control over pitching, fielding, base-running, and a number of other tactics common to the game of baseball. Active participation is the key in this exciting and challenging baseball simulation. The game may be played by persons of all ages. It is available on cassette only in either BASIC (C1020) or Extended BASIC (C1020X).

Extended Baseball was developed by Extended Software Company and requires Extended BASIC. The user controls the pitcher and batter, as well as a number of functions such as balls, strikes, inning changes, scoring, hitting, fielding, etc. You are also given such statistics as batting averages. The game is available on disk or cassette.

All*Star Bowling is a simulated bowling game, which may be played by as many as eight players at the same time. You may throw the ball and knock down the pins at various speeds. Before rolling the ball you must position it in the lane. This game is designed for use with Extended BASIC and was developed by Ehninger Associates, Inc. It is available on cassette only (C1110X).

Decathlon, developed by Pewterware, is based on the Olympic Decathlon event. The user is required to compete in all ten events, and timing is very important. You have one second to complete the first event, with the time increasing in one-second increments with each event, so the final event must be completed in ten seconds. This is an exciting and challenging game designed for users aged 10 and older. The game is available on cassette only (CPW67030).

Football has been designed by TI for users 8 years old and up. It provides a simulation of football based on actual pro football statistics. The player can select offensive plays, defensive plays, and acts as the quarterback on offense.

Indoor Soccer is a five-man soccer game. The user controls the players and can make passes, shots, interceptions, saves, tackles, and use a number of other common tactics. This is a fast-paced game. One of the unique features of this game is that you can view an instant replay of each score. The game is designed for users 8 and older and is available on module only for $30.00 (PHM 3024) from TI.

GAMES OF SKILL

In **The Attack**, the user is the commander of a spaceship and must destroy enemies. The user must maneuver his ship to avoid contact with alien ships and fire missiles at the same time. Developed by Milton Bradley Company, this is an exciting and entertaining game, which can be played by users of all ages. It is available in plug-in module form only and is priced at about $40.00 (PHM 3031).

Blasto can be played by one or two players. It is a tank game in which the user must destroy a mine field while avoiding opponent fire at the same time. This is a fast-paced game, which is timed, so you must quickly destroy as many mines as possible. A number of options are provided, and there are dangers involved. If you hit a mine at close range, for example, you must start over. This game is designed for users 10 or older and is available as a plug-in module for about $25.00 (PHM 3032).

In **Cars and Carcasses**, developed by Not-Polyoptics, you are the driver of a car on a randomly generated board. The object of the game is to save an imaginary city from monsters such as Frankensteins, Draculas, and weird space creatures, by running them over as you move around the board. This exciting and challenging adventure game is available on cassette only.

Car Wars is a speed-racing game of skill in which the player is pitted against the computer. The object is to maneuver your car around the track while avoiding obstacles. This game has several difficulty levels and may be used with the TI wired remote controllers if desired. It is available as a plug-in module and sells for about $40.00

Chutes & Sharks, also developed by Ehninger Associates, Inc., is designed for single-player use. In this game you are in control of a boat waiting for paratroopers who are dropped from a helicopter positioned overhead. To further complicate the situation, there are sharks in the waters waiting to destroy the paratroopers if you do not position the boat to receive the falling paratroopers properly. This program requires Extended BASIC and the memory expansion unit may also be used, although it is not required. The game is available on cassette only (C1120XM).

Galactic Gunfight, developed by Intersoft, puts you in control of a spaceship that must defend a colony against these invaders, which appear on the screen in squadrons of five at increasing speed levels. The game is designed for users 7 and older and is available on cassette (CIS64540).

Hustle is a fast-paced game designed for one or two players. The user is in control of a snake-like object with which he must try to hit targets while avoiding his opponent, the edge of the screen, and his own object, at the same time. This is an excellent tool for developing quick reflexes, as well as eye and hand coordination. It can be used by users 10 and older and is available in module form only for about $25.00 (PHM 3034).

Ships!, another adventure game from Not-Polyoptics, can be played by one or two players. You are in command of a ship on the high seas and must steer your ship in a variety of changing conditions, such as wind changes, which make sailing more difficult. The graphics in this game are good. The game is available on cassette.

Speedway 100 is a well-illustrated graphic game in which the player controls one of six cars on a speedway. He first selects the number of laps he wishes to make and then maneuvers his car around all other cars, which appear in different lanes at varying speeds. Other obstacles are provided to increase driving proficiency. The game is available on cassette only, and was developed by Intersoft (CIS 64540).

Tickworld may be unpleasant for some; you are pitted against eight gigantic, hungry ticks. This game creates a different game board each time it is played, and your job is to capture the ticks and imprison them before they get you. There are three skill levels. This game was developed by Not-Polyoptics and is available on cassette.

TI Invaders is a one-player game in which you are attacked by strange space creatures. You must be quick to use your missiles before these multi-color creatures get you. The wired remote controllers from TI are optional, and the game comes in plug-in module form for approximately $40.00.

TI-Trek is an exciting and challenging game that uses the speech capabilities of the TI-99/4A. You are responsible for the safety of a galaxy and have the ability to fire phasors, torpedoes, or multiple torpedoes in an effort to destroy the enemy before he endangers your galaxy. A warp control is provided for additional maneuvers. The package is available as a plug-in module for $15.00 (PHD 5002).

Tournament Brick Bat was designed by Image Producers and pits the player against the computer or a human opponent in a fast-action skill game. The computer keeps a record of the score and the game increases in difficulty as skills improve. Designed for users 10 and older, the game is available on cassette only (9041).

Video Games I consists of three games: Pot-Shot, Pinball, and Doodle. Each is designed for use by persons of all ages. Pot-Shot is an aim-practicing game, Pinball is designed in the format of the pinball games found in arcades, and Doodle is a game in which the user tries to trap his opponent. Each game is sure to provide many hours of entertainment. Available as a plug-in module, it is priced at $30.00 (PHM 3018).

In **ZeroZap**, the user is provided with a computerized pinball game that includes electric lights and fascinating sound effects. The user may also create an individualized playing field, providing an educational as well as entertaining game. This program was developed by Milton Bradley Company and may be used by children as young as 6. Available as a plug-in module, it sells for $20.00 (PHM 3036).

GAMES OF STRATEGY AND LOGIC

Advance is a board game developed by Not-Polyoptics. In this strategy game, two or three players compete in moving up the board. You can purchase squares on the board, and the whole game provides a number of random selections. Each square has a point value, or it may take points away from the player after purchase. Blocking other players stops them from reaching the end. This game is available on cassette only.

A-Maze-Ing is an exciting and challenging game of mazes. The user is able to select from options providing many mazes, from the very simple to the very complex. The chances of seeing the same maze twice are small—this game provides 5,200 variations. The package is available as a plug-in module and sells for $25.00 (PHM 3030).

Strategy is used to the fullest in **Barrier.** Two players are required; the first player tries to draw a continuous line on the screen from left to right, while the second does the same from top to bottom. The object is to reach your goal in the shortest time possible, while preventing your opponent from reaching his goal. The game is designed for persons of all ages and is available on cassette only.

Brain Games was developed by Creative Computing. It consists of five challenging games designed to test your brain power. The games are Dueling Digits, Parrot, Tunnel Vision, European Maps, and U.S. Maps. All games are designed for users of all ages, and the package is available on cassette (CS-6002).

Challenge I displays ten frogs on the screen. The user is prompted to use logic to take the frogs through leaps that will reverse the color pattern of the frogs. This strategy game is designed for use by two players and was developed by Ehninger Associates, Inc. It is available on cassette only (C1030).

Challenge II contains two programs. The first is NIM, a game two players may play. Objects are arranged in rows and the players begin removing the objects, one at a time, until the winner is determined. The winner is the one who removes the final object. The second game is the popular game of Tic-Tac-Toe. Designed by Ehninger Associates, Inc., Challenge II is available on cassette only (C1040).

Connect Four was developed by Milton Bradley Company and is the computerized version of the popular game. In this strategy game the user has to place four markers in a row (down, across, or diagonally) to win. The game may be played by persons 10 and older and is priced at $20.00. It is available as a plug-in module only (PHM 3038).

Corner Bound is a combination skill and strategy game in which a snake-like line is presented on the screen. You are in control of the line, and you must maneuver it to hit targets placed in the corners of the screen. This is an excellent teaching aid, using both eye and hand coordination, and three skill levels are provided to increase your proficiency. The game was developed by Microcomputers Corporation and can be played by persons aged 8 and older. It is available on disk (MCD0001) or cassette (MCT0001).

Crosses is a computer simulation of a combination of Go and Othello, two popular board games. This game is presented in board form, and two players are required to strategically compete, placing markers on the board and trying to capture the opponent's marker while making a cross on the board. This game was developed by Not-Polyoptics, and is available on cassette only.

The Cube was developed by Linear Aesthetic Systems, and provides a graphic simulation of Rubik's Cube. High resolution graphics make this game exciting visually and challenging as well. You have complete control over the movements of the cube and can command it to display any of six sides, rotate it clockwise or counterclockwise, and spin it to see another side, etc. This skill-challenger is available on cassette only.

Doctor Nuttier was developed by Ehninger Associates, Inc. It is more an entertainment program than game. Dr. Nuttier is the computer, and the player is prompted to enter a question. Using psychoanalysis techniques developed by Carl Rogers, the doctor provides advice based upon the input. The program is available on disk only (D1050).

Hangman was developed by Hall Software. Two players are required; one selects a word, and the other must guess the word before his man is hung. You have only seven guesses in this game. It is available on cassette only.

Hangman is another computerized version of the popular game. The computer selects a mystery word, or the user can enter his own word. An opponent is asked to guess the letters in the mystery word. Each incorrect guess causes another portion of the hanged man's body to be drawn on the gallows. The object is to guess the word before the man is hanged. Designed by Milton Bradley Company, this package sells for $20.00 and can be played by users 6 years and older. It is available as a plug-in module (PHM 3037).

Extended Hangman is yet another computerized version of this popular game. This package was developed by Extended Software Company and requires Extended BASIC for operation. It uses color, graphics, and speech, making this extended form of Hangman unique and entertaining. Over 500 words are included in the computer's word vocabulary, and you can add your own words to this list to make the game more difficult or easier. The game is available on disk or cassette.

Hidden Numbers was developed by Hall Software. It is designed to test the memory skills of the player. Numbers are hidden behind several rows of squares. The player must locate the squares hiding matching numbers. The game may be played by persons of all ages and is available on cassette only.

Hunt the Wumpus is a search for the Wumpus through a hidden maze of caverns and tunnels. Clues are provided, and the user must evaluate the clues and avoid any dangers while traveling through the maze. This program is available in module form only and sells for $25.00 (PHM 3023).

Match Wits is a game that tests the concentration of the player. The game is designed for up to four players aged 16 and older and is available on cassette only (CPW67050). It was developed by Pewterware.

Mind Challengers includes two challenging games. The first game is similar to many hand-held games available commercially and prompts the user to repeat a sequence of notes. A second user is then prompted to repeat the previous sequence plus an additional note. The game continues until one player is unable to echo the correct sequence, up to 64 notes. The second game is a code-breaking game, which uses colors and shapes. Both games can be used by persons 10 and older. It is available in plug-in module form for $25.00 (PHM 3025).

Mind Masters, developed by Image Producers, is a strategy and logic-teaching simulation game in which the computer creates problems and the user must solve them. The game is designed for multi-player use, and different skill levels may be used by each player. Here, the principles of deductive logic are taught in a fun and challenging manner that tests the user's skill, ability, and patience. The package is designed for persons 10 and older and is available only on cassette (9405).

In **Mystery Melody**, a challenging musical game, the user and his opponent are prompted to guess the title of a song based on notes provided. The winner is the person able to name the song in the fewest guesses. One person may play this game alone, or it can be played by the entire family. It will provide hours of entertainment. It is available on cassette (PHT 6010) or disk (PHD 5010) for $10.00 and $15.00, respectively.

Oldies But Goodies—Games I is five games in one. It includes Number Scramble, Word Scramble, Tic-Tac-Toe, Biorhythm and Factor Foe. Each game may be played with the computer or with a human component. The package is designed for use by persons of all ages. It can be purchased on cassette (PHT 6015) or disk (PHD 5015) for $15.00 and $20.00, respectively.

Oldies But Goodies—Games II contains five games. Hammurabi, Hidden Paris, Peg Jump, 3D Tic-Tac-Toe, and Word Safari. Designed for all family members, it is available on cassette (PHT 6017) for $20.00 or on disk (PHD 5017) for approximately $25.00.

Othello™ is the computerized version of this popular board game. Each of the two players take turns placing disks on the board in an attempt to bracket the opponent's disks and thus make them his own. The winner is the player with the most disks on the board at the end of the game. The game is over when neither player is able to make a move. Alternately, one player can be pitted against the computer. This game is available in plug-in module form only (PHM 3067) for $40.00.

Peg Jump, developed by Hall Software, simulates the popular pegboard game in computerized form. You are required to make carefully planned jumps to win this fastpaced game. The game is available on cassette only.

Saturday Night Bingo is a computer simulation of that highly popular game, Bingo. It is a multi-player game in which the computer randomly selects numbers and then reads them out loud through the TI speech synthesizer, an optional peripheral. Two modes are provided, automatic and manual, so the user may select the speed at which the game is played. This package may be used at home with the entire family, or it may be used by church and other types of organizations that stage Bingo games.

The game is available on cassette (PHT 6025) for approximately $25.00 or disk (PHD 5025) for $30.00.

Scrambled Letters Puzzle & Number and Alphabet Hi-Lo is a two-game package developed by Hall Software. In the first game you are required to unscramble 15 letters. In the second game, Number and Alphabet Hi-Lo, a character is randomly selected by the computer, and you are prompted to guess the number or letter. The computer responds to your guesses in a manner which gives you clues as to how close you are to guessing the correct number or letter. This package is available on cassette only.

Skill Builder I, designed by Image Producers, consists of two games of skill at different skill levels. The first is Bingo Duel, in which one or two players are provided with problems to solve. In the second game, Number Hunt, the user must match numbers at increasing levels of difficulty. Designed for users 10 and older, the package is available on cassette only (9406).

Strategy Games was developed by Creative Computing to promote reasoning and strategy skills. Four games are included in this package, including Blockade, Checkers, Darts, and Depth Charge. This package is available on cassette (CS-6003).

Strategy and Brain Games, also developed by Creative Computing, combines the programs included in the Brain and Strategy packages previously discussed. This package is available only on disk (CS-6501).

Strategy Pack I contains two strategy games in which the user can play against the computer or a human opponent. The first is Roman Checkers, based on the popular board game of the same name. The second game is called Frame Up. In this game you must use strategy and skill to outwit either the computer or your human opponent. The package was designed by Image Producers and is available on cassette only (9404).

Video Chess, developed with the help of International Master David Levy, is designed for chess players of all ages. It is simple to use, providing help with moves if desired. You can play with the computer or another person, and different levels of play are provided. If desired, a game can be stopped and stored for return to the same game at another time. The computer keeps track of each move, and although it is simple to use, will prove challenging to even the most experienced chess player. The cost is approximately $70.00, and is available from Texas Instruments in module form (PHM 3008).

Wall Street—a Market Simulation is a computerized simulation of the stock market, in which you must make as much money as possible. You are given ten years in which to make your fortune. Although advertised as a game, this package will provide education in money management and the stock market strategies. It may be used with TI BASIC or Extended BASIC and is available on cassette only.

Wall Street Challenge, developed by Image Producers, is a computer simulation of Wall Street in which the user is allowed to invest in different types of stocks. Charts and a Dow Jones report are provided to keep you up to date on the current trends. The package is designed for persons aged 13 and older and is available on cassette only (9402).

Wildcatting was developed by Image

Producers. In this game, the computer sets up hidden oil deposits, and the user must try to locate them. Geological survey data is provided in order to help you determine whether or not oil is in a given location, and the oil deposits are in different locations each time the game is played. Designed for persons 10 and older, Wildcatting is an educational game that will teach the user logical decision-making. It is available on cassette only (9403).

Yahtzee, developed by Milton Bradley Company, is a challenging dice game that many people are familiar with in its uncomputerized format. Points are garnered by varying dice combinations, and the winner is the person who obtains the most points after a specific number of rolls. Designed for users 8 years old and older, this package is available as a module and sells for $25.00 (PHM 3039).

FANTASY AND ADVENTURE

Adventure is a series of games developed by Adventure International. Each require the adventure command module, the TI disk memory system for the disk version, or a cassette recorder and the TI cassette interface cable for the cassette version. These games were developed to create fantasies, which may take as long as a few weeks to complete. The games include Mystery Fun House, Ghost Town, Adventureland, and a host of others. The disk version (PHM3041D) is available for $50.00, as is the cassette version (PHM3041T).

Adventureland Adventure Database: the player in this game is taken on a fantasy trip to a forest in an enchanted world. You must explore the world, searching for treasures and avoiding all obstacles. You are also required to locate the secret place where the treasures are stored. The game is available on disk (PHD 5046) or cassette (PHT 6046) and requires the adventure command module from Texas Instruments. It is priced at $30.00 for either cassette or disk.

Alpiner is a mountain-climbing game, which may be played by one or two players. You are presented with a number of obstacles during your climb up a choice of mountains, including Matterhorn, Kenya, McKinley, Garmo, Everest, and Hood. Dangers you must confront include the abominable snowman, lions, bears, skunks, forest fires, avalanches, and rockfalls. The game is colorful and has sound effects as well. Alpiner is available as a plug-in module (PHM 3056) for approximately $40.00.

Airmail Pilot takes you back to the early days of aviation. You are the pilot, and you are given the responsibility of piloting a plane from Columbus to Chicago in the shortest time possible. Many factors are involved in this flight, however, to make it challenging and educational. You must contend with weather conditions, electrical storms, etc. The game was developed by Instant Software and is designed for persons 10 and older. It is available on cassette only (0274 TI).

In **Chisolm Trail** the user is in control of a steer which must move through a series of mazes, avoiding obstacles and killing monsters that block the way. Chisolm Trail may be used with the TI optional joysticks and comes in plug-in module form (PHM 3110). Price is about $40.00.

The Count Adventure Database requires the adventure command module, and sets you back in the days of Dracula in Transylvania. You are given a number of clues and then must determine who you are, what you are doing in Transylvania, and a number of other things. The game is available on disk (PHD 5049) or cassette (PHT 6049) for $30.00.

Galactic War is a space game in which you are in control of a spaceship. You must avoid all obstacles and prevent any danger to your spaceship while trying to destroy all enemy spaceships. This game requires Extended BASIC and is available on cassette (X1100X). The package was designed by Ehninger Associates, Inc.

Ghost Town is a treasure-hunting game placed in a setting of an old ghost town. The player must go through the deserted buildings, looking for treasure and avoiding ghosts. This game requires the adventure command module from TI, as well as a disk drive and controller if purchased on disk (PHD 5053) or a cassette recorder and cable if purchased on cassette (PHT 6053). It is priced at $30.00 for either disk or cassette.

The Golden Voyage Adventure Database sends you back in time to a royal palace in a Persian City. You are introduced to a very old king, who is dying. Your mission is to find a way to restore his youth, equipped with only a bag of gold. Your quest is to find the Fountain of Youth before the king dies. This adventure requires the adventure command module from TI and can be purchased on disk (PHD 5056) or cassette (PHT 6056) for approximately $30.00.

Gorfia Pestulitas is an unusual adventure game that places you in outer space and pits you against alien ships constantly trying to attack you. You may also opt to have space mines in the game, and two skill levels are provided. This game requires Extended BASIC and was developed by Extended Software Company. It comes on disk or cassette.

Khe Sanh is a game in which you are in Vietnam and must defend your base against the enemy. This tactical skill game pits you against approaching forces. It is your job to locate the enemy by means of search and destroy tactics, as well as defend approaching supply convoys, and defoliate forests, etc. This game was developed by Not-Polyoptics and can be purchased on cassette only.

Maze of Ariel is a game which displays random mazes consisting of different rooms and paths through them. You are pitted against the computer, and to create difficulty there is a dragon that you must avoid. You are equipped with only a flashlight and a supply of grenades. Developed by Not-Polyoptics, the game comes on cassette only.

Mission Impossible Adventure Database is loosely based on the popular television program of the same name, which held many persons spellbound during its long run. As the game begins, you are listening to a recording in a briefing room. Your mission is to locate a person who has set out to destroy a nuclear reactor, thus destroying the entire world. This action-packed game must be used with the adventure command module and is available on disk (PHD 5047) or cassette (PHT 6047) for approximately $30.00.

In the **Mystery Fun House Adventure Database** the player must get inside the fun house, which may not be very easy. Once inside, you are confronted with all the sights you would see in a fun house, and you must search for a prize hidden somewhere inside. The game requires the adventure command module and is available on disk (PHD 5051) or cassette (PHT 6051) for approximately $30.00.

In **Parsec**, you are the commander of a spaceship in outer space. You are required to do battle with alien ships, which attack in varying patterns, while guiding your ship through refueling tunnels. At different levels of the game, you are attacked by different types of alien ships, and as the levels change, you must guide your ship through obstacles such as asteroid belts. A number of controls are at your disposal, such as speed/sensitivity control, pause capability, and even a female voice, which can apprise you of your current situation. The graphics and speech capabilities make this game quite exciting and entertaining. Joysticks and a speech synthesizer are optional for this game, which is available in plug-in module form only (PHM 3112) and sells for $40.00.

In **Pyramid of Doom Adventure Database** the player is positioned in a desert. Sticking out of the sand is a pole, which marks the point where a pyramid has been discovered. It is your job to find the entrance to the interior of the pyramid, find and collect the treasures, and then escape. This is an exciting adventure game, which requires the adventure command module. It is available on disk (PHD 5052) or cassette (PHT 6042) for $30.00.

SAM (surface-to-air missile) Defense is a simulation game. In this game you are the viewer, watching the firing of a surface-to-air missile. This package is available on cassette only (C1080), and was developed by Ehninger Associates, Inc.

Santa Paravia and Fiumaccio was designed by Instant Software. In this game, up to six players can compete to become the ruler of a medieval state (king or queen). Various situations are set up and the players must create a kingdom. This game may be played by persons 12 and older. It is available on cassette only (0273 TI).

Savage Island I and II Adventure Database requires the adventure command module. This game places you on the edge of a wild and impenetrable jungle. You are required to enter the jungle, with all its creepy creatures, and travel through it successfully. This is a two-part game, which is available on disk (PHD 5054) or cassette (PHT 6054) for approximately $40.00.

Sengoku Jidai is a fantasy game set in medieval Japan. Designed for one to three players, you are given a castle and three armies consisting of archers, foot soldiers, and samurai. You must then defend your castle as well as attack the enemy. Each mapboard is randomly selected by the computer, which makes this adventure game different each time it is played. This package was designed by Not-Polyoptics, a division of Synchronet, and is available on cassette.

Starship Pegasus is a space adventure game. This game is designed for play by one person only, and provides a different set of randomly selected circumstances each time it is played. You are positioned inside a spaceship, looking out at the solar system. You are provided with indications of what is occurring by sensors, which tell you conditions of a planet, for example. The purpose is to conquer the planet or solar system, while avoiding space pirates. Judgment decisions are involved, as well as skill and coordination. This game was developed by Not-Polyoptics and is available on cassette.

3-D Star Trek is a three-dimensional computerized version of Star Trek. It is a very challenging and exciting program package, designed for persons 16 and older. Available on cassette, this game was designed by Color Software.

In **Strange Odyssey Adventure Database** the player finds himself stranded on a small planet with a spaceship badly in need of repair. The object is to find the parts needed to repair the ship, while collecting treasures from the ancient civilization's ruins. The game requires the adventure command module and can be purchased on disk (PHD 5050) or cassette (PHT 6050) for $30.00.

In **Tombstone City: 21st Century** you are in an Old West ghost town that is being invaded by strange alien creatures, which eat people and tumbleweeds. You are given a security force consisting of prairie schooners, and it is your responsibility to protect the ghost town from the aliens. This is a one-player game, which will test skill and strategy. It may be used with the optional wired remote controllers from Texas Instruments and is available as a plug-in module only (PHM 3052) for $40.00.

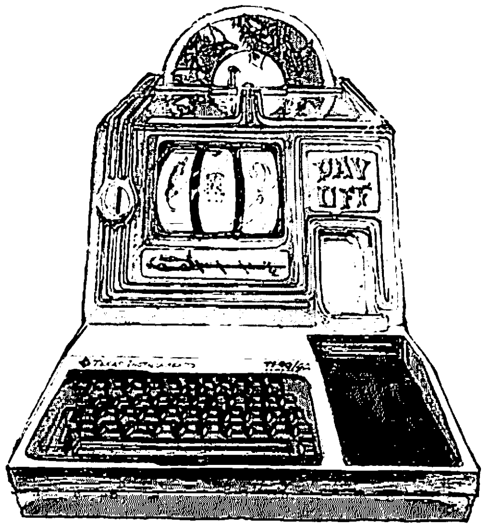
In **Trail West** you are required to travel west to California to reach the gold mines. Many obstacles confront you on this 2,000-mile trip, and you are required to reach your destination without running out of ammunition or supplies. Random events such as storms and overturned wagons are introduced during the trip, and you must use skill and logic to ration supplies and ammunition so they will last the entire trip. The game can be played by young and old and was developed by Micro-Ed, Inc. It is available on disk only (OT-1).

Treasure Dive, developed by Tutorex, takes the player through a search for sunken treasure. You are a scuba diver and are presented with many obstacles such as sea monsters and a limited air supply. Users of all ages will enjoy this game, which is available on cassette only (TUT 6861).

Tunnels of Doom is a fantasy/adventure game, which takes you back to the age of kings and queens in a search for treasure. This is a role-playing game in which you have many options to choose from while you attempt to rescue the king and defend yourself against monsters and other dangers. This game requires the disk drive and controller if purchased on a disk, or a cassette recorder and cable if purchased on cassette. It is priced at $60.00 for disk (PHM 3042-D) or cassette (PHM 3042-T).

In Voodoo Castle Adventure Database you are presented with a closed coffin, and it is your job to locate the information needed to free the Count from a curse that has placed him in the coffin. The game requires the adventure command module and may be purchased on disk (PHD 5048) or cassette (PHT 6048) for $30.00.

Chapter 6



TI BASIC Conversions

While there are many software companies which offer products for the TI-99/4A, you will not find a large number of books, such as this one, which include programs written specifically for the TI-99/4A. Texas Instruments is the largest producer of software for this machine, and you can find just about anything you want. However, you will still want to write your own programs and probably run programs that were originally written for other machines.

There are many books out which include program line listings for the TRS-80, IBM Personal Computer, Apple, ATARI, Commodore, and many others. In many instances, it is possible to convert these programs so that they will run in TI BASIC on the TI-99/4A computer. Such conversions are often not too difficult when the programs deal with the handling of on-screen text. However, computer games, especially those which involve on-screen animation, may present some serious conversion difficulties. You will often run into situations where a true conversion is not possible. Rather than converting a program written for another computer, you simply have to write a program for the TI-99/4A that will duplicate the on-screen action of the original program. Converting a program involves leaving most program lines intact, but changing many of the statements to conform to the new dialect of BASIC. For example, the CLS (clear screen) statement, which is common to most dialects of BASIC, is not found in TI BASIC. To convert a program that includes CLS, all that is necessary is to replace this statement with call clear, which does the same thing in TI BASIC. This is a program conversion. However, some programs may contain statements that perform actions that are simply not available with a similar set of statements on the TI-99/4A computer. Here, it may be necessary to develop an entirely new set of instructions that will perform the same on-screen functions using the TI-99/4A as the previous program segment did for the other machine. This is not a conversion, but a program section rewrite.

The following discussion is an overview of statement, command, and function modifications that may be necessary to get a program written for another machine to run on the TI-99/4A. It certainly doesn't take every other dialect of BASIC into account, but hopefully, this information will be a worthwhile guideline as to how to proceed.

ABS—ABS is the absolute value function. It is always used in the same manner from dialect to dialect of BASIC. If you see this function in a BASIC program written for another computer, no modifications should be necessary in order to get the program to run on the TI-99/4A.

ATN—The same is true of the ATN function, which returns the arc tangent of an argument. I have never seen it used in a different manner than that specified in the Texas Instruments manual.

ASC—The ASC function will give you the ASCII character code that corresponds to the first letter in a string argument. This function is used in basically the same manner in all dialects of BASIC.

Break—This command will not be encountered in other dialects of BASIC, although it is similar to the stop and wait commands found on other machines. When converting programs from other dialects to TI BASIC, you will probably never have to use the break command or the unbreak command.

Bye—I have not encountered the bye in other dialects of BASIC. However, it is equivalent to the new command found in TRS-80 BASIC and IBM BASIC. In these two machines, BASIC is exited and a return to the main operating system occurs. Since this is a command, it should not be encountered in any BASIC program lines.

Call CHAR—Call CHAR is a subprogram found in TI BASIC and in no other dialects. When involved in graphics programming, you will encounter so many differences between dialects that it is probably much easier to simply write a new program from scratch rather than to attempt a conversion. The Call CHAR command is used to create block characters. This feature may not be available on other machines, at least not through a single command.

Call Clear—This is identical to the CLS command, which is found in many dialects of BASIC. It may be used either as a command or a statement. When it is found in a program, however, it is a statement. Call CLEAR may be directly substituted for CLS when converting programs to TI BASIC.

Call Color—This is equivalent to the color statement found in other dialects of BASIC. The numerical commands which follow color may vary from machine to machine, depending on the number of foreground and background colors offered. You will have to do a bit of experimenting when attempting a conversion using this statement. However, it should be done only after the major portion of the program has been modified and is running. Color statements are used primarily in graphics, and again, it may be easier to simply rewrite the program rather than convert.

Call VCHAR/Call HCHAR—These two subprograms in TI BASIC do not really correspond to any other statements in other dialects of BASIC. They are, however, roughly equivalent to locate statements found in IBM BASIC and TRS-80 Color Computer Extended BASIC, or to print @ statements in TRS-80 BASIC or Apple BASIC. In these dialects, however, the statements are used to print information at a specific point on the screen, whereas in TI BASIC, the subprograms are more often associated with graphics rather than text. In most applications involving a printout of text mode information, locate statements can be omitted from your TI version of the program and print @ statements can be replaced with print. These statements are often used to display information at certain points on the screen to make the output more readable. In this case, the omission of locate or the changing of print @ to print will not make a significant difference. However, if these statements are used to form charts, a conversion may not be possible.

Call JOYST—This is the joystick subprogram in TI BASIC. It may correspond roughly to the stick function found in some dialects of BASIC.

Call Key—The Call Key subprogram is quite similar to the on key and key on statements in other dialects of BASIC. These statements are used to activate a certain key or set of keys on the keyboard and create branches when the key is activated.

Call Screen—This subprogram is used primarily in graphics programming. It loosely corresponds to screen statements in other dialects, but the numbers used to designate foreground and background colors will probably be different. In most instances, the Call Screen subprogram can be used to replace a screen statement, but you will have to coordinate the numbers between the dialects. In IBM BASIC, for example, the screen statement is used to establish screen mode, which has to do with screen width and resolution.

Call Sound—This statement corresponds to the sound statement as well as the beep and play statements in other dialects. The Call Sound subprogram may be used to directly replace beep, although you will have to follow Call Sound with appropriate numbers to produce a 1000 Hertz tone for a short duration. The sound statement in IBM BASIC can also be converted to Call Sound, although the duration portion must be altered to reflect TI BASIC nomenclature. In IBM BASIC, duration is given in machine cycles, of which there are approximately 18 per second. There is no volume command, but this will be of little significance in making the conversion to TI BASIC.

CHR$—This function returns the character corresponding to any given ASCII character code and is common to all dialects of BASIC. However, different machines will have different character sets, so it may be necessary to compare the character sets and ASCII codes in order to determine which character is being specified by an ASCII number. For example, in IBM BASIC, ASCII code 219 is a block character used in text mode graphics. There is no equivalent to this character in the TI-99 character set. In such instances, a conversion may not be possible.

Close—This statement is used to discontinue access to a file. In many instances, it can be used interchangeably from dialect to dialect.

Continue—Continue is a command in TI BASIC, so it should not be encountered in a program line listing. However, it may be necessary to continue execution after a program halt has been performed. In TI BASIC, you may use CON or continue. Both are the equivalent of CONT, which will be found in some other dialects of BASIC.

COS—This function returns a cosine of an argument. It is used in the same way in all common dialects of BASIC.

DATA—The data statement is common to all dialects of BASIC and need not be altered when making a conversion.

DEF—The DEF statement is used to define your own function to be used in a program. Most dialects of BASIC contain a DEF FN statement, which is directly interchangeable with DEF in TI BASIC.

Delete—In TI BASIC the delete command is used to erase a program or data file from a disk. It will not be found in a program line listing. However, its use in TI BASIC is quite different from other dialects. In TRS-80 and IBM BASIC, the delete command is used to delete lines from a program in memory. In many other dialects, the kill command is used to erase programs from disks, and this command most closely corresponds to the delete command in TI BASIC.

DIM—The dimension statement is common to all dialects of BASIC. No changes should be necessary to make a conversion.

Display—The display statement will probably not be encountered in other dialects of BASIC. It is rarely used in TI BASIC, as the print statement is more commonly used to perform the same function.

Edit—This command is used to display a line for editing purposes. Most machines have a similar command. It is never encountered in a program line listing.

End—This statement stops program execution and is the same in all dialects of BASIC.

EOF—The EOF function indicates an end-of-file condition. It is common to many other dialects, so no changes will be necessary as long as you use the number specified in all other open and close statements as an argument in the EOF functions.

EXP—The exponential function is the same in most dialects of BASIC.

For-To-Step—This statement is common to all dialects of BASIC.

GOSUB/GOTO—These are branch statements, which are common to most dialects of BASIC.

If-Then-Else—This statement is common to most dialects of BASIC, but there are subtle differences from dialect to dialect. In TI BASIC, this statement must be followed by a branch line. In other dialects, if-then may be followed by statements or commands, as in IF A = 6 THEN PRINT "HELLO". This line would not be legal in TI BASIC. The modification would have to be:

```
10  IF A = 6 THEN 20 Else 30
20  PRINT "HELLO"
30  END
```

If-then-else statements in other dialects of BASIC may also include Boolean operators, such as AND, OR, etc., as in:

```
10  IF A = 6 AND B = 10 THEN 30
    ELSE 20
20  END
30  PRINT "HELLO"
```

In TI BASIC, this would have to be modified to:

```
10  IF A = 6 THEN 15 ELSE 20
15  IF B = 10 THEN 30 ELSE 20
20  END
30  PRINT "HELLO"
```

As indicated by these modifications, all if-then-else statements in other dialects of BASIC can be converted to run in TI BASIC, although more lines will usually be required. In TI Extended BASIC, however, statements may follow if-then-else statements, and Boolean operators are allowed.

Input—The input statement is common to most dialects of BASIC, although the formats may vary. TI BASIC requires that a colon be used following any printed phrase in an input statement. For example,

```
10 INPUT "PRESS ENTER TO CON-
   TINUE":A$
```

In other dialects, the colon is rarely used. Instead, you will find a comma or a semicolon. This punctuation change is necessary only when a quoted phrase follows the input statement. Such program lines as INPUT A, INPUT A$, etc. need no changes at all to run in TI BASIC.

INT—This is the integer function, which normally requires no modifications to run in TI BASIC.

Len—The len function is common to most dialects of BASIC and determines the number of characters in a string statement. No modification is usually required.

Let—The let statement is optional in TI BASIC, as in some other dialects. Let statements need no modifications to run in TI BASIC, nor is it necessary to add them when modifying programs from dialects where let is not used.

List—The list command is common to most dialects of BASIC and is used to list the lines of the program currently in memory. It will not be encountered in a program line listing.

Log—The Log function returns the natural logarithm of a number. It needs no modification during a conversion to TI BASIC.

New—The new command is used to erase the current program from memory in order to clear space for the writing of a new program. It is common to most dialects of BASIC and will not be encountered in a program line listing.

Next—See For-To-Step.

Number—This command, which may also be entered as NUM in TI BASIC, is used to automatically generate line numbers each time the enter key is pressed when you are writing or entering a program. Although some machines do not offer this feature, those that do may use NUMS. Numbers, and AUTO.

Old—This command loads a program from cassette into current memory. Most often, it corresponds to the load command found in other dialects of BASIC.

ON-GOSUB/ON-GOTO—Both of these statements are common to other dialects of BASIC. See also GOSUB/GOTO.

Open—The open statement is used to open a file and usually does not need to be changed when converting a program to TI BASIC.

Print—All dialects of BASIC contain a print statement, which is used to display information on the monitor screen. Some modifications may be necessary. For instance, the following lines are legal in TI BASIC:

```
PRINT A
PRINT A$
PRINT "HELLO",A
PRINT "HELLO";A
PRINT A,"HELLO"
PRINT A;"HELLO"
```

The following lines may be legal in other dialects, but not in TI BASIC:

```
PRINT A"HELLO"
PRINT "HELLO" A
PRINT "HELLO" A "HELLO AGAIN"
```

Whenever a variable is used on the same line with a quoted phrase, the variable must be separated from the phrase by either a comma or a semicolon. Many other dialects of BASIC do not require this.

Randomize—When a conversion to TI BASIC is done, randomize can be used to directly replace the randomize statements in the original programs. Some dialects require that a number follow the randomize statement, such as RANDOMIZE 32, or RANDOMIZE X, where X is an assigned numeric variable. In most instances, you can replace these expressions with the single randomize statement in TI BASIC.

Read—The read statement is common to all dialects of BASIC and can usually be used as it is.

REM—REM statements are nonexecutable and may be used in any dialect of BASIC.

Resequence—Resequence or RES is a command that is used to sequentially renumber the lines in a computer program. It is equivalent to RENUM in some dialects of BASIC.

Restore—The restore statement is the same in most dialects of BASIC and is used to enable read statements to reaccess the first item in a data line.

Return—This statement is always used in conjunction with the GOSUB statement, both of which are common to most dialects of BASIC.

RND—The random function is used in most dialects of BASIC, but some machines randomize in slightly different manners. For example in one dialect of BASIC

```
INT (RND*6)
```

will return a number of from 1 to 6. To accomplish this in TI BASIC, the line must be modified to:

```
X = INT(RND*6) + 1
```

Run—The run command is common to most dialects of BASIC.

Save—The save command is used to copy the current program in memory on to a tape or disk. This command is used similarly in most dialects of BASIC.

**SEG$**—In TI BASIC, the SEG$ function takes the place of LEFT$, RIGHT$, and MID$

in some other dialects of BASIC. Any of these three functions can be replaced by SEG$. You will rarely encounter SEG$ in other dialects of BASIC.

SGN—This function gives the algebraic sign (plus, minus, zero) of an argument. It may be used in the same manner in most dialects of BASIC.

SIN—This function returns the sine of a number and need not be changed.

SQR—This function returns the square root of a number and is used in the same manner in most other dialects.

Stop—This statement is common to most dialects of BASIC. No modifications should be necessary when it is encountered in a program line listing.

**STR$**—STR$ is used to return a string representation of the value of an argument. It need not be changed during a conversion to TI BASIC.

Tab—Most dialects of BASIC use the tab function. However, you will need to know the differences among screen formats of the various machines. The TI-99/4A is capable of a 32-column screen, but other machines may have 40 or 80 column capability. Thus, a TAB(35) would be illegal on the TI-99/4A, but would be legal on a machine with the capability of displaying 35 columns or more.

TAN—This function returns the tangent of a number and is used in most dialects of BASIC.

VAL—The VAL function converts a string variable to a numeric variable and can be used in most dialects of BASIC.

MULTIPLE STATEMENTS

The previous material has presented the various statements, commands, and functions and compared their usage in TI BASIC to their usage in other BASIC dialects. However, it is also important to know that TI BASIC does not allow multiple statements on a single program line, and modifications will be necessary when they are encountered in a program that you wish to convert to run on the TI-99/4A.

Multiple statement lines may be presented in different ways from machine to machine. For example, one machine may use the following format:

```
10 LET A = 10/PRINT A
```

Here, two statements have been included in a single line separated by a backslash (/). In other dialects of BASIC, the same operation would be presented as follows:

```
10 LET A = 10:PRINT A
```

Here, a colon is used to separate the statements. Other dialects may use a semicolon or even brackets. The first statement assigns a value of 10 to the variable A. The second prints the value of A on the display screen. In order to modify this line to run on the TI-99/4A, the following would be necessary:

```
10  LET A = 10
20  PRINT A
```

All multiple statement lines cannot be handled in the manner already discussed. For example, lines containing if-then-else statements will be a bit more difficult to separate, as in the following program section:

```
10  IF A = 10 THEN PRINT
    "HELLO": PRINT "YELLOW"
20  PRINT "GOODBYE"
30  END
```

You might think that a conversion to TI BASIC would look like this:

```
10  IF A = 10 THEN 15 ELSE 20
15  PRINT "HELLO"
20  PRINT "YELLOW"
30  PRINT "GOODBYE"
40  END
```

This is not correct, however. Line 10 contains the if-then statement. Therefore, the second statement on that line (PRINT "YELLOW") will be executed only if A is equal to 10. In the modified program, PRINT "YELLOW" will occur whether or not A is equal to 10. Thus, the correct modification would be:

```
10  IF A = 10 THEN 15 ELSE 25
15  PRINT "HELLO"
20  PRINT "YELLOW"
25  PRINT "GOODBYE"
30  END
```

In this program, if A equals 10, "HELLO" will be printed on the screen. If A is not equal to 10, there is a branch to line 25, where the word "GOODBYE" will be printed and the program will end. This problem is encountered because both the if-then statement and the second statement on the same line must be modified.

If you stick to text mode programs written in other dialects of BASIC for your first exercises in program conversion, you will learn how to do such conversions easily.

Later when you try the more difficult graphics programs, you can concentrate on the more involved areas. As you become more familiar with the TI-99/4A, you will become more adept at making conversions as you learn the programming methods used to accomplish certain functions. You will then be able to look at a program and duplicate the functions on your machine. TI Extended BASIC, which is more powerful than TI BASIC, will make conversion much more simple, because it contains many programming features common to other dialects of BASIC.

Appendix A
ASCII Character Codes

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
32	(space)	48	0	64	@ (at sign)
33	! (exclamation point)	49	1	65	A
34	" (quote)	50	2	66	B
35	# (number or pound sign)	51	3	67	C
36	$ (dollar)	52	4	68	D
37	% (percent)	53	5	69	E
38	& (ampersand)	54	6	70	F
39	' (apostrophe)	55	7	71	G
40	((open parenthesis)	56	8	72	H
41	) (close parenthesis)	57	9	73	I
42	* (asterisk)	58	: (colon)	74	J
43	+ (plus)	59	; (semicolon)	75	K
44	, (comma)	60	< (less than)	76	L
45	– (minus)	61	= (equals)	77	M
46	. (period)	62	> (greater than)	78	N
47	/ (slant)	63	? (question mark)	79	O

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
80	P	97	A	114	R
81	Q	98	B	115	S
82	R	99	C	116	T
83	S	100	D	117	U
84	T	101	E	118	V
85	U	102	F	119	W
86	V	103	G	120	X
87	W	104	H	121	Y
88	X	105	I	122	Z
89	Y	106	J	123	{ (left brace)
90	Z	107	K	124	:
91	[(open bracket)	108	L	125	} (right brace)
92	\ (reverse slant)	109	M	126	~ (tilde)
93	] (close bracket)	110	N	127	DEL (appears on screen as a blank.)
94	∧ (exponentiation)	111	O		
95	________ (line)	112	P		
96	(grave)	113	Q		

Appendix B

A Concise Guide to TI-99/4A BASIC

The language used by most microcomputers is BASIC, an acronym for *B*eginners *A*ll Purpose *S*ymbolic *I*nstruction *C*ode. Unlike many computer languages, BASIC uses English words to represent computer commands. For example, the print statement tells the computer to print something on the screen. The end statement tells the computer to stop the execution of a program. The BASIC commands, statements, and functions relate to the actual function that is to be carried out.

If you're already familiar with BASIC, you will have little trouble converting to TI BASIC. All dialects of BASIC are similar, although some contain special statements designed to perform a specialized function on a particular machine. These differences are always minor, and most of what you already know about BASIC will apply to the TI-99/4A.

This chapter overviews TI BASIC and explains what each command, statement, and function causes the machine to do. If you're familiar with BASIC, many of these pages will contain review material; otherwise, this chapter will serve as a BASIC primer for the TI-99/4A.

The nucleus of TI-99/4A BASIC is built into the machine. The BASIC interpreter is written into the on-board ROM contained in the console unit. (ROM stands for Read-Only Memory, as opposed to RAM, which is Random-Access Memory.) The programs contained in ROM are handled on the machine level: the integrated circuit chips that make up ROM have been electronically programmed at the factory. When the computer is turned on, it reads this information into its microprocessor. Nothing you can do at the keyboard affects the programming in ROM.

The programs you write are committed to RAM. RAM is also composed of integrated circuits, but you can change this programming based on your keyboard input. RAM is also known as read/write memory: you can write information into the memory and then the microprocessor reads information out. The language used to write information into RAM is the one that is set up in ROM.

The language set in ROM is much like a dictionary, which contains all the words in the English language. In dictionary form the words are not connected to form meaningful sentences, and this is the way the words are organized in ROM. ROM simply provides you with words you can use. You must pull them out and arrange them in a meaningful order, which will then be committed to RAM as a program.

Each TI BASIC statement, command, and function, including what it means and how to use it in writing programs is explained below.

ABS The absolute value function gives the absolute value of an expression. This expression is often called the argument; it is the value obtained when the numeric expression is evaluated. If the argument is positive, the absolute value function gives you the argument itself. If the argument is negative, the absolute value is the negative of the argument (the absolute value of −20 is 20). This function is useful when it is necessary to pull the absolute value from a long series of mathematical functions. It is used in the following format:

```
ABS(38)
```

The 38 in this case is the numeric expression. It could be replaced by a variable or a complex series of expressions, such as:

```
ABS 20*(14*3.2)/-20)
```

The absolute value will return the numerical value from this formula and delete the minus sign if the value is negative.

ASC The ASC function returns the ASCII code for the first character of a string variable or string of numbers inserted in parentheses following this function. Each character produced by the TI-99 is represented and accessed by an ASCII code number. For example, the ASCII code number for the uppercase letter O is 79. Using the ASC function followed by an O in parentheses would yield the number 79. A typical format for this function is:

```
10 X$ = "O"
20 PRINT ASC(X$)
```

When this simple program is run, the computer screen will display 79 (the ASCII value for X$), which is equal to the uppercase letter O.

ATN This function returns the arc tangent of the numeric expression that follows it in parentheses. The arc tangent is the angle in radians whose tangent is equal to the numeric expression. (This sophisticated mathematical function will not be of immediate use to the beginning programmer.) ATN function formatting is handled in the same manner as the ABS function.

Break The break command is entered via the keyboard: it is not normally included as part of a program. When the break command is entered, break points are set at the program lines currently being executed. When you enter break, you tell the computer to stop running the program before executing the statement on the next line.

BYE The BYE command lets you leave BASIC. When this command is entered, the computer closes all open files; the program in memory and all variables are erased; and the computer is reset so it's ready to receive programming when you return to BASIC. After the BYE command is entered and executed, the computer screen returns to the master mode, the first mode accessed when the computer is turned on. Don't execute this command until you are certain that any program currently in memory has been saved.

**CHR$** The CHR$ function is the reverse of the ASC function. Where the ASC function returned the ASCII code for a specific character, the CHR$ function converts an ASCII code number into its character equivalent. The following will cause the computer screen to display the letter O:

```
10 V$ = CHR$(79)
20 PRINT V$
```

In the simple program shown here, V$ equals CHR$(79), which is the same as saying V$ is equal to ASCII character 79 or the uppercase letter O. The print statement in line 20 causes the character O to be printed on the screen.

Close The close statement "closes" a file that was previously opened using an open statement. Any open file must be closed before the computer can move to another part of the program. The close statement is discussed further under the open statement entry.

Call CHAR Call CHAR is a subprogram standing for character definition. The call statement is used to call up or initiate the subprogram. Call CHAR lets you arrange special graphics characters on the screen. It is followed by the ASCII character code and a pattern identifier expressed in hexadecimal code (a 16-character string expression which specifies the pattern of a character you want to use in your program). The graphics section of this book discusses this in more detail.

Call Clear The Call Clear subprogram clears or erases the monitor screen. A call clear command is often issued at the beginning of a program to clear the screen.

Call Color The Call Color subprogram lets you specify the colors of characters on the screen. This subprogram statement is followed by a character set number, foreground color code, and background color code, all numeric expressions.

Call GCHAR This subprogram lets you read a character anywhere on the screen, by specifying the row number, column number, and the numeric variable to read the character. The video screen is arranged in a series of blocks, 32 running horizontally and 24 running vertically. Row number 12 references the middle far left of the screen, while column number 16 reference's the top center portion of the screen. When the two numbers are combined, as in 12,16 the center of the screen is referenced.

Call HCHAR This subprogram places a character anywhere on the screen and optionally repeats it horizontally. To use this program, you must input the row and column numbers, along with the character code (given in the ASCII equivalent) and, optionally, the number of repetitions.

Call JOYST This subprogram lets you input information directly to the computer by positioning the lever on a joystick. (Joysticks are available as options for the TI-99.)

Call Key The Call Key subprogram transfers one character from the keyboard directly to the program, eliminating the need for an input statement. (The Call Key subprogram is similar to an INKEY$ variable common to other dialects of BASIC.) This subprogram reads the keyboard input and branches the program according to the pressed key.

Call Screen This subprogram is used to display on-screen graphics and lets the screen be changed to any of 16 available colors. When a Call Screen subprogram is executed, only the screen background color changes. The Call Screen color code is a number from 1 to 16. To change a screen to a dark blue background, you would type CALL SCREEN(5) (5 is the color code for dark blue).

Call Sound The Call Sound subprogram generates tones and noises. This statement must be followed by the time duration, frequency, and volume you wish the sound to follow. The duration is measured in milliseconds, numerically expressed by a value of from 1 to 4250. A value of 4250 holds the tone for 4.25 seconds. Frequency is expressed in hertz; legal values are from 110 to 44,733. A chart in the Appendices indicates the frequencies that correspond to different musical notes. The final number in the string expresses volume, one of five values from 0 to 5. Zero is the loudest; 5 is the softest.

Call VCHAR This subprogram is like Call HCHAR, except it repeats characters on the screen vertically rather than horizontally.

To further demonstrate, consider the following example:

```
CALL HCHAR(2,15,72,7)
```

This will cause 7 ASCII characters (86) to appear vertically on the screen starting at position 2,15. ASCII character 86 is the capital letter V, which is repeated 7 times. The HCHAR version is:

```
CALL HCHAR(2,15,72,7)
```

The ASCII code has been changed to 72, the letter H, which will be printed 7 times horizontally.

Continue This command is entered whenever program execution has been halted by a break command. When a continue command is input, execution continues until the program ends or another break point is reached.

COS The cosine function returns the cosine of a numeric expression. The format is COS(X), where X is the numeric expression. If you entered the line PRINT COS(4), the screen would display the cosine of the number 4. You can also use this function as follows:

```
10  I = COS(4)
20  PRINT I
```

Data The data statement stores numeric and string constant data in a program. It is always used with a read statement, which instructs the computer to pull information from the data statement. The format for the data statement is: Data item, item, item, . . . The items must be separated by commas. If you wanted to include the numbers 1 through 10 in a data statement you would use DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Whenever a read statement is encountered, the information contained in the data statement will be fed to the machine one item at a time. A read statement would have to be accessed 10 times to read all data items in the example.

DEF The define statement lets you define your own functions in a particular program. The specified function name may be any valid variable name. Any parameters following a DEF statement must be enclosed in parentheses.

Delete This command removes a program or a data file from a disk. To use this command, you must have the TI Disk Drive Controller and a disk drive. Once a file is established, the delete command will erase it from the storage medium. The command must be followed by the file name or program name. If you opened a file under the name GAME, DELETE "GAME" will erase it from the disk.

DIM This may be used as a command or statement and reserves space for numeric and string arrays. DIM lets you set the maximum size of an array. For example DIM X(15) sets aside a one-dimensional array with a maximum of 15 elements. Using the DIM statement, you may also establish two- and three-dimensional arrays.

Display The display statement is identical to the print statement. Both may be used to write information on the display screen. The display statement causes information to be output only to the screen.

Edit The edit command is entered in direct mode and used to call up a line from a previously written program to change it. For example, to make corrections in line 100, input EDIT 100, and that line will appear on the screen. The FCTN and cursor movement keys are used to align the cursor with the beginning of the word or letter to be changed. New information may now be typed over the old, or the insert function may be used to place letters or words before this point. There is no need to retype the entire line. The line number cannot be changed. Press enter to exit the edit mode and store all changes in memory.

End The end statement terminates your program. It may be used interchangeably with the stop statement. Its presence as the last line of a program is not necessary, since the program will automatically terminate when there are no more lines to execute. The end command is useful when one or more subroutines are included at a point in the program that follows the normal termination point. For example, if you write a program filling lines 100 through 1000 and then add a subroutine starting at line 1010 reached through a GOSUB or GOTO statement, the end statement might be inserted at line 1005 to avoid accidental execution of the subroutine at the end of the program.

EOF The end-of-file function determines when the end of a specific file has been reached. When files are accessed by the open statement, their information is output until there is nothing left. On the next information loop, an end-of-file condition results. Using the EOF function, a branch may be built into a file-reading program that will terminate the program before an error message can occur or activate other programs. If a file has been opened as number 1, the EOF function might look like this:

```
IF EOF(1) THEN 1000
```

When an end-of-file condition results in file number 1, the program will branch to line 1000.

EXP This is the exponential function, the inverse of the natural logarithm function. It raises the number 2.718281828 to the *X* power. In this case, the variable *X* is the number you input. For example:

```
PRINT EXP(4)
```

will raise the number 2.718281828 to the fourth power.

For-to-Step This statement is used to create loops in a computer program. It is always used with a next statement, which marks the end of a loop. While for and to must always be used to set up a for-next loop, the step command is necessary only when the loop is to cycle in increments other than 1. The following program demonstrates the use of this statement:

```
10  FOR X = 1 TO 10 STEP 1
20  PRINT X
30  NEXT X
```

This is a simple for-next loop that causes the value of *X* to be printed on the screen. The for-to-step statement in line 10 specifies that *X* is assigned values of from 1 to 10 in steps of 1. In this loop, *X* is equal to 1 on the first cycle, 2 on the next, then 3, and so on, until the maximum value specified is reached. If the step were changed to 2, the count would skip every other number.

GOSUB The GOSUB statement is used to branch to another portion of a program. It may be typed as one word or two, as in GOSUB or GO SUB. This statement is always used with a return statement allowing you to defer the program to a subroutine, execute each line in the subroutine, and then return to the next program line following the GOSUB statement.

GOTO The GOTO statement like the GOSUB statement, is used to branch from one portion of a program to another. A line number indicating the program line to which to branch follows this statement. GOTO 100 or GO TO 100 will bring about a branch to line 100. Once a GOTO branch is made, there is no automatic return; the only way to return to the main program is with another GOTO statement.

If-Then-Else This statement lets you change the sequence of your program execution by using a conditional branch. GOSUB or GOTO will bring about an unconditional branch. With if-then-else, a certain condition must exist before the branch occurs. Else is often dropped from this statement. For example:

```
IF X = 40 THEN 500
```

This means there will be a branch to line 500 only when the value of *X* is equal to 40. If *X* is not equal to 40, the computer will execute the next line. When the else statement is used, a branch will always occur, but the branch selected depends on a certain condition:

```
IF X = 40 THEN 500 ELSE 1000
```

There are two possible branches—one to

line 500 and the other to line 1000. If the value of *X* is 40, there will be a branch to line 500; if *X* is not equal to 40, then there will be a branch to line 1000. If-then-else statements are often used to conditionally access subroutines using branches to lines containing GOSUB statements. The following program segment demonstrates this:

```
10  IF X = 1 THEN 20 ELSE 30
20  GOSUB 100
30  PRINT X
40  END
```

In line 10, the computer is told to branch to line 20 if the value of *X* is 1. Line 20 contains a GOSUB statement that branches to a subroutine starting at line 100. The content of this subroutine is unimportant for this discussion and is therefore not included here, but when it has been executed, there will be a return statement that will cause line 30 to be the first executed after the subroutine.

If *X* is not equal to 1, the else portion of line 10 branches to line 30, skipping line 20 altogether.

Input The input statement temporarily halts program execution until information can be input via the keyboard. The input prompt appears as a question mark on the screen. The input statement may be immediately followed by a prompt message in quotation marks. After the last quotation mark, a colon must be inserted and then a variable name. Either a numeric or string variable may be specified. If a numeric variable is used and the information is not input in numeric form, an error message will be displayed.

Another form of the input statement lets you enter data from an accessory device. The input statement can be used only with files opened in the input or update mode. The file number in the input statement must be the file number of a currently open file.

INT The integer function gives you the largest integer not greater than the argument. The argument is the value obtained when a numeric expression is evaluated. With positive numbers, the decimal portion of the number is dropped. For negative numbers, the next smallest integer value is used. The following format is used with the INT function:

```
100  X = INT (113.876)
110  PRINT X
```

The value output to the screen will be 113, the integer of 113.876. If the value in line 100 was $-L$113.876, the integer value would be −114, the next smaller integer value. INT is used whenever it is necessary to arrive at answers given as whole numbers only and not as whole numbers and fractions of their decimal equivalents.

LEN The length function gives you the number of characters in a string:

```
10 A$ = "HELLO"
20 PRINT LEN(A$)
```

When the program is run, the screen will display the number 5, which is the number of characters in A$ (HELLO). The LEN function counts spaces as well as characters; if A$ were assigned the value of HELLO CATHY, the length would be 11.

Let The let statement is optional and is used to assign values to variables within a program. Because let is optional in TI BASIC LET A = 10 and A = 10 are both acceptable.

List The list command is entered in direct mode (no line number) rather than as part of the program. It causes the screen to display the list of lines that make up a program. You may also specify the name of the device on which you want the lines listed. You can specify a line or lines with the list statement; typing LIST displays all program lines. LIST 150 will display only line 150. LIST—150 will list all program lines before and including line 150. LIST 150– will list line 150 and all lines following it. LIST 90—150 will list lines 90 through line 150.

LOG This is the natural logarithm function. PRINT LOG(3.5) will give you the natural logarithm of the number 3.5. The number may also be represented by a previously assigned variable. LOG is the inverse of the EXP function.

New The new command erases the program currently in memory. It also closes any open files and clears all space previously allocated for special characters. The new command is often used after a program has been written, debugged, and stored on cassette or disk. Typing NEW erases any program from memory and lets you begin on a new one. Don't use new before a program you wish to save has been committed to permanent storage.

Next The next statement is never used by itself; it is always paired with a for statement (for-to-step). The next statement controls whether the computer will repeat a loop or exit to the following program line. When a next statement is encountered, the previously evaluated increment in the step clause is added to the control variable, and then the control variable is tested to see if it exceeds the previously established limit.

Number This command may be entered as NUMBER or NUM. When the computer receives this command, it automatically generates line numbers to speed program writing. The NUM command is issued before a program is written. When you press enter, a line number is automatically generated, starting with 100 and stepping up in increments of 10. When you have finished the program, hit enter once more to remove the number generator feature.

Old The old command reads a previously saved program into the computer's memory. This applies to programs that have been saved on cassette or disk and then removed from current memory. When you want to load information from cassette to current memory, input OLD CS1. A set of instructions will then appear on the screen telling you to rewind the cassette tape and press the cassette play button. The old command is followed by the name of the file you wish to load into memory.

ON-GOSUB The ON-GOSUB statement is used to tell the computer to perform one of several subroutines. It is another way of setting up a conditional branch to subroutines without using the if-then-else statement.

```
10 INPUT A
20 ON A GOSUB 150,250,350
```

This is not a conditional branch in the true sense, but it does bring about several branches whenever a value is input for A. The following is an example of a true conditional branch:

```
10  B = 10
20  INPUT A
30  ON B-1 GOSUB 1000
```

Here, a branch will occur only when A is equal to 9 (the value of B minus 1, as specified in the ON-GOSUB statement).

ON-GOTO Like ON/GOSUB, this statement is used to access different portions of a program, whereas with GOTO returns are unnecessary.

Open The open statement prepares a BASIC program to use data files stored on cassette, disk, etc. It provides the necessary link between a file number in a program and the particular accessory device on which the file is to be located.

Option Base The option base statement is used to set the lower limits of an array subscript at 1 instead of 0.

POS The position function detects the occurrence of a substring within a string. The POS function compares two strings and indicates at what position a letter or series of letters found in one string begins occurring in another.

Print The print statement is used to display information on the screen. Words to be displayed follow the print statement in quotation marks. Words and/or numbers assigned to string variables or variables are printed by following the print statement with the name of the variable without quotation marks.

Randomize The randomize statement is used with the random number function (RND) to generate a pseudo-random sequence of numbers. When randomize is used by itself, the random number function generates a different sequence of random numbers each time. The randomize statement may also be used with another number called the seed. If the seed is unchanged, the sequence of random numbers will be the same each time the program is run. The randomize statement and RND function are used in programs simulating dice rolls, card selections, and other games of chance. The output numbers are pseudo-random, which means they are logical progressions to the computer, but the patterns are so complex as to appear random to users.

Read The read statement is used to read data from data statements within the same program. Read and data statements must always be used together. Information is read an item at a time, sequentially from left to right.

REM The REM (remark) statement is a non-executable portion of a program. The computer simply skips over the lines that begin with REM. REM statements are used to insert information concerning the program that may be of importance to users.

REM statements are also helpful to the programmer. For example, when programs are quite long and complex, the beginning and ending of certain subroutines can be identified with REM statements, as can other major building block programs within a major program. When the program is reviewed for debugging, the REM statements let you quickly identify the sections you seek.

Resequence The resequence command may also be entered as RES. It reassigns the line numbers for all lines in a program. It is often necessary during debugging to insert additional program lines, making the line number sequence confusing.

Also, when many additional lines must be input, you can often run out of space between lines. The resequence command, when used alone, will automatically renumber every line in steps of 10, beginning with line number 100. All branch statements are also automatically changed to reflect the new numbers: if one branch statement was input as GOTO 100 and line 100 was changed to line 120, during resequencing, the GOTO statement would read GOTO 120 after resequencing. You can also renumber a program starting at certain lines and determine your own sequence: RESEQUENCE 10,10 causes the first resequenced line number to be 10, followed by 20, 30, 40, etc.

Restore The restore statement is used to return a data statement to the beginning of its list of items.

Return The return statement is used with a GOSUB statement to return program execution to the line immediately following the GOSUB statement that accessed the subroutine.

RND This is the random function, which provides the next pseudo-random number in the current sequence of numbers generated by the randomize statement.

Run The run command is used to begin execution of a program in memory. When used by itself, the run command causes execution to begin at the first line number in the program. If the run command is followed by a line number, execution will begin at that line.

Save The save command lets you copy the current program in memory onto disk or cassette. The save command must be followed by the name of the file you wish to establish.

SEG$ This is a function which gives you a portion of a designated string. The following program demonstrates the use of this function:

```
10  A$ = "PARADOXICAL"
20  PRINT SEG$(A$,4,5)
```

When this program is run, the screen will display ADOXI. This is the segment of the word "paradoxical" that begins with the fourth letter from the left and continues for five letters. SEG$ has been used to extract a substring from A$.

SGN This is the signum function, giving you the algebraic sign of a value specified by an argument. This function tells you whether a number is positive, negative, or equal to 0:

```
10  A = 15
20  PRINT SGN(A)
```

Here a 1 will be displayed on the screen, indicating that the number is positive. If A were changed to −15 in line 10, the screen would display −1, incidating that the value of A is negative. If A were equal to 0, 0 would appear on the screen. Obviously, in the program shown for demonstration purposes, it is quite easy to tell whether a number is positive, negative, or equal to 0. The SGN function, however, may be used in a program that performs complex mathematical functions, most of which are not displayed on the screen.

Here, the SGN function may also be used to bring about branches to other portions of the program:

```
IF SGN(A) = -1 THEN 200
```

The value of the number is unimportant; the important quality is whether it is negative rather than positive or equal to 0.

SIN The sine function gives you the trigonometric sine of the argument. If the angle is in degrees, multiply the degrees by pi divided by 180 to get the equivalent angle in radians. The SIN function is useful when performing different types of vector math and in generating sine waves on a computer screen graph.

SQR The square root function returns the positive square root of the value specified by the argument:

```
10 PRINT SQR(9)
```

The output from this program will be the number 3, which is the square root of 9.

Stop The stop statement terminates a program and is interchangeable with end.

STR$ This function converts the number specified by an argument into a string. It is the opposite of the VAL function.

TAB The tab function is used with the print statement and specifies a starting position on the line for the next print item. The tab function works like a tab on a typewriter. PRINT TAB(10);"HELLO" will print HELLO on the screen starting 10 positions from the left.

TAN This returns the tangent of the argument X, where X is an angle in radians.

Trace This lets you see the order in which the lines of your program are executed. When the trace command is input, the line numbers appear on the screen as they are executed. This can be a most valuable debugging aid, in that infinite loops and unwanted branches can be quickly detected. To remove the trace feature, the untrace command is input.

VAL The VAL function is used to extract a numeric value from a string variable. If the string variable is composed of numbers only, the VAL statement will extract these and assign them to a numeric variable that may be used in mathematical functions. When VAL is used with a string variable containing letters and numbers, only the numeric portion will be committed to a numeric variable.

Glossary

accessory device—Any equipment that attaches to the computer to allow it to expand its functions. Accessory devices are usually limited to hardware or firmware, as opposed to software.

access time—The interval between the application of an input pulse and the availability of data signals at the output.

algorithm—An algorithm is a set of rules or a standard procedure that provides the solution to a problem. In a computer program, an algorithm is the most efficient method of achieving a specific goal. In this case, efficient would most likely refer to a minimum number of statements, functions, and commands or the shortest program possible to achieve the goal.

alphanumeric—Alphanumeric describes characters which include the letters of the alphabet, numerals, and symbols used for punctuation and mathematical operations.

array—An array is a group or table of values referenced by the same name when programming in BASIC. Each item or value in the array is often referred to as an element. Array elements are variables and can be used in expressions and in BASIC statements or functions that allow the use of variables.

ASCII—ASCII is an acronym for American Standard Code for Information Interchange. This is an 8-level code (7 bits plus parity check) that is widely used for information interchange. This code structure is used in the TI-99/4A and most personal computers to represent letters, numbers, symbols, and special characters.

assigned statement—An assigned statement is a line in a computer program that assigns a value of an expression to a variable. On the TI-99/4A the let statement or the equal sign may be used.

asynchronous—Asynchronous is a mode of computer operation in which performance of the next command is activated by a signal indicating that the previous command has been completed.

BASIC—An acronym for Beginners All Purpose Symbolic Instruction Code, BASIC is a programming language that is used to write programs. A BASIC program consists of one or more statements, functions, or commands preceded by line numbers. These numbers control the sequence in which the instructions are run. The BASIC language used with all computers is quite similar. Different types of computers may alter the BASIC language slightly to conform to certain machine standards. The TI-99/4A is programmed in TI BASIC.

binary—Binary is a number system based on only two digits, 0 and 1. The internal language and operations of digital computers are most often based on the binary system.

bit—Bit is an abbreviation for binary digit. This is an information unit equal to one binary decision, or the designation of one or two possible values. These values may be referred to as high/low, 1/0, yes/no, off/on, etc.

Boolean algebra—Boolean algebra is a deductive system or process of reasoning named after George Boole, an English mathematician. It is a system of theorems that uses symbolic logic to denote classes of elements, true or false propositions, and on-off logic circuit elements. Symbols are used to represent operators such as and, or, not, except, if-then, etc. This system is now recognized as an effective method of handling single-valued functions with two possible output states. When Boolean algebra is applied to binary arithmetic, the two states become 0 and 1. When applied to switching theory, the two states become open and closed.

branch—A branch is a break in the sequential execution of a program. A branch causes the computer to jump to another portion of the program. In TI BASIC, statements that set up branches include GOTO, GOSUB, and if-then-else. There are two types of branches, conditional and unconditional. An unconditional branch is conducted each time the line that includes the branch instruction is executed. A conditional branch brings about the jump only upon the result of some logical or arithmetic operation. GOTO and GOSUB statements are used most often to bring about unconditional branches, and if-then-else is used only for conditional branching.

breakpoint—In TI BASIC, a breakpoint is the point in a program at which execution is halted by the break command. After execution has been suspended, you can perform operations in command mode to help locate program errors. To resume execution after a breakpoint, type CONTINUE and press the enter key.

buffer—A buffer is a device or area of computer memory that serves as an isolator or interface to dissimilar elements. In computer terminology, a buffer is usually thought of as a storage device. It may store input or output information transmitted at one rate until another station can use the data. The output from the Texas Instruments computer to the printer is transmitted at a much faster speed than the printer can transfer to paper. The print buffer receives the output from the computer at its normally transmitted rate. It stores the information until the printer can accept it all at its own speed.

bug—A bug is an error. The term applies especially to software errors. When a program is first written, it must often go through a debugging process. This is a matter of removing all errors.

bus—A bus is a conductor through which information is transmitted or received.

byte—A byte is a string of binary digits that form one unit. A byte is equal to one character letter, number, space, or punctuation mark. Computer memory capacity is specified in bytes. The TI-99/4A makes available 16,000 bytes of read/write memory (RAM). This may also be specified as 16K (RAM). This may also be specified as 16K bytes, with the K meaning multiply by 1000.

card—In microcomputer jargon, a card is a plug-in circuit board. The peripheral expansion system used with the TI-99/4A makes available slots for inserting these circuit boards, or cards. The plug-in modules contain internal cards, but since they are enclosed in one unit, the term module applies.

cassette storage system—This system allows you to to write information onto a blank cassette tape and to load the program or data back into the computer.

cathode ray tube—Abbreviated CRT, a cathode-ray tube is a device that displays information. Your television picture tube is a cathode ray tube, and all monitors, such as the TI 10-Inch Color Monitor, contain them.

character—A character is a letter, number, or symbol that can be produced (usually on the screen) by a computer.

character set—A character set is a set of representations, called characters, from which selections are made to denote and distinguish data. A set may include the numerals 0 to 9, the letters A to Z, punctuation marks, and a blank or space.

chip—A chip is a thin piece of silicon material. Solid state devices use a single chip to produce highly complex circuits, all contained on the chip surface. More common terminology lets chip be used to describe integrated circuits.

code—A code is a system of symbols for representing data or instructions in computers. Code also means to translate a program into instructions acceptable to a particular computer.

collate—An operation in which two or more sets of data are merged to produce one or more sets that still reflect the original ordering relations.

command—A command is an instruction that is not a part of a program and which the computer can perform immediately upon input.

command module—The Texas Instruments command modules are preprogrammed read-only memory circuits enclosed in a package for insertion in the TI-99/4A console. These modules contain different languages and programs.

concatenation—Concatenation is the process of linking together in a series.

cursor—A cursor is a symbol that appears on the monitor to indicate where the next character will appear. In TI BASIC, the cursor is represented by the more than symbol (>).

data—Data are facts, concepts, numbers, letters, symbols, or instructions for communication, interpretation, or processing. Data form the basic elements of information. (See data statement)

data statement—In TI BASIC, a data statement allows for items to be contained within a program line. A matching read statement reads each data item when commanded to do so. A data statement may contain as many constants as will fit on a program line, and any number of data statements may be used within a program. The items contained in data statements are always read sequentially.

debug—Debug is a computer term used to describe the detecting and removing of errors and malfunctions from a program or from the computer itself.

default—A default is a characteristic or value that the computer assumes to be true unless otherwise negated.

digital system—A digital system is a device or circuit that deals in digital rather than analog form. It operates on a binary number configuration using 2 as a base and the digits 0 and 1 as values, which are referred to as bits. Combinations of these bit values provide the code by which data can be processed through its electronic circuitry.

DIP—DIP is an abbreviation for dual in-line package. This describes many types of integrated circuit packaging. DIP packages resemble long, flat wafers with pins extruding the longer edges.

directory—A directory is the area on a disk in which the names of files are stored. Included in a directory may be information about the size of the file, its location on the disk, and the date it was created.

disk—A disk (also called a floppy disk) is a mass storage device. It is a flexible circular object coated with a magnetic substance. When in use, the disk spins inside a permanent protective jacket. The disk drive contains a magnetic head to read and write information from the disk.

edit—Editing involves the deletion, insertion, and rearrangement of data.

error message—An error message is a screen prompt that appears after you enter a line incorrectly, or when you are trying to run a program with incorrect lines. The error message indicates what the problem may be and may generate an error message number for easy reference in the TI reference manual.

executable statement—Executable statements are program instructions that tell

BASIC what to do while executing a program.

execute—Execute means to run a program.

exponent—An exponent is a number which indicates the power to which another number or expression is to be raised. In TI BASIC, the exponent is written to the right of the number or expression, separated from it by the > symbol which is typed in via the keyboard.

expression—an expression is a combination of variables, constants, and operators that can be evaluated to a single result.

file—A file is a collection of information usually stored on cassette tape or disk.

flowchart—A flowchart is a graphical representation of the definition or solution of a problem, in which symbols are used to represent functions, operations, and execution flow. A flowchart contains the logical steps in a program so the designer can conceptualize and visualize each step. It defines the major phases of the processing, as well as the path to problem solution.

formatting—Formatting is the process of setting up a disk to receive information. This process checks the disk for bad spots and builds a directory to hold information about the files that will be written on it.

function—A function is a feature that lets you specify a variety of procedures as single operations. Each procedure may actually contain a large number of steps.

function keys—On the TI-99/4A, the keyboard function keys perform special functions when pressed simultaneously with the FCTN key. These functions can include the printing of quotation marks in a program, or the deletion, insertion, or complete erasure of characters or lines in a program while in the edit mode.

graphics—Graphics include simple drawings, random patterns, and graphs.

hardware—Hardware refers to the physical components that make up the microcomputer. A monitor, printer, cassette recorder, and disk drive, are hardware.

hexadecimal—Hexadecimal describes a number system that has a base of 16 and uses 16 symbols. These symbols are the numbers 0 through 9 and the letters A through F.

housecleaning—Housecleaning is a process by which BASIC collects all of its useful data and frees up areas of memory that were once used for strings. The data is compressed so the user can continue until there is no space left.

instruction—An instruction defines an operation and causes the computer to actually perform the operation.

integer—An integer is a positive or negative whole number, such as 1, 2, 3, 4, or 5. Zero is an integer, but numbers such as 1.12, 2.333, and 4.115 are not.

integrated circuit—Abbreviated IC, an integrated circuit is an interconnected array of components fabricated from a single crystal of semiconductor material (usually treated silicon).

interface—An interface lets a computer operate into a communications line, a terminal, or into peripheral devices.

I/O—I/O is an abbreviation for input/output. An I/O channel is a circuit path that allows communications between the processor and devices including the keyboard, a disk drive, a cassette player, etc.

iteration—A programming technique of repeating a group of program statements. For-next loops are often used for this purpose.

joystick—A joystick is a lever that provides coordinate data of a display surface. The data can control operations, such as the movement of one or more display elements. Joysticks are often used in computer games or to manipulate data on the screen.

keyboard—The keyboard contains keys for entering data or information into the system.

light pen—A light pen is a photosensitive device that causes the computer to modify the display on the monitor screen. The light pen signals the computer using an electronically produced pulse. The light pen can draw impressions on the monitor screen, as well as read points of light from computer-generated displays.

logical operator—Logical operators perform logical, or Boolean, operations on numeric values. Logical operators are usually used to connect two or more relations and return a true or false value to be used in a decision. A logical operator takes a combination of true-false values and returns a true or false result. An operand of a logical operator is considered to be true if it is not equal to zero, or false if it is equal to zero. The result of the logical operation is a number that is true if it is not equal to zero, or false if it is equal to zero. The number is calculated by performing the operation bit by bit. The logical operators are not (logical complement), and (conjunction), or (disjunction), XOR (exclusive OR), IMP (implication), and EQV (equivalence).

logic expression—A logic expression consists of variable array elements, function references, logic constants, and combinations of operands separated by logical operators and parentheses. Typically, logical expressions may contain arithmetic expressions separated by relational operators.

loop—A loop is the repeated execution of a series of instructions usually for a specific number of times. For-next statements are often used to establish such loops.

machine language—A machine language is used directly by a microprocessor. All other languages must be translated or compiled into binary code before entering the processor.

matrix printer—A matrix printer is a device that uses an array of dots to form characters.

memory—Memory in a computer stores information. Random-access memory (RAM) is the memory section to which operator programs are written and stored for execution. Read-only memory (ROM) is not accessible, having been programmed at the factory to allow the computer to perform its built-in functions.

microprocessor—A microprocessor is a central processing unit.

modem—A modem is an electronic device that performs the modulation and demodulation functions required for communications. A modem can be used to connect computers and terminals over telephone circuits.

module—A module is an assembly which contains a complete circuit or subcircuit. Printed circuit boards designed to be plugged into a computer may be classified as modules.

monitor—A monitor is a unit in a computer that prepares machine instructions from a source code. It may use built-in compilers for one or more program languages. The machine instructions are sequences into the processing unit once compiling is complete. A monitor most often refers to the display screen.

nanosecond—A nanosecond is an amount of time equal to 10^{-9} second. It is abbreviated ns and is equivalent to 1/1,000,000 of a second. A time interval of 1,000,000 nanoseconds is equal to 1 second.

non-executable statement—A non-executable statement does not cause any program action. The statements are there within the program, but they are passed over and not acted on. Two examples of non-executable statements are REM and data.

null string—A string that has no value is a null string. It contains no characters and has a length of 0.

number mode—This is an automatic line numbering mode set up by the NUM command in TI BASIC. Once this command is executed, a number will be generated on the screen whenever the enter key is pressed.

numeric comparison—In numeric comparison, when arithmetic and relational operators are combined in one expression, the arithmetic is always performed first. For example, the expression:

$$X + Y < (T-1)/Z$$

will be true if the value of X plus Y is less than the value of T−1 divided by Z.

numeric expression—A numeric expression may be a numeric constant or variable, or may be used to combine constants and variables using operators to produce a single numeric value. Numeric operators perform mathematical or logical operations mostly on numeric values, and sometimes on string values. They are referred to as numeric operators because they produce a value that is a number.

numeric function—A function is used to call a predetermined operation that is to be performed on one or more operands. BASIC has intrinsic functions that reside in the system, such as SQR (square root) or SIN (sine).

operator—An operator is a symbol used in performing arithmetic calculations. In the calculation 1 + 1 = 2, the operator is +. Common operators are *, +, −, /,∧. These stand for multiply, add, subtract, divide, and raise to.

power supply—A power supply is an electric circuit that supplies operating voltage and current to the computer.

program—A program is a set of instructions that direct a computer in performing a desired operation, such as the solution of a mathematical problem or the sorting of data.

program line—Any line preceded by a line number and containing a statement is called a program line.

prompt—A symbol or phrase that appears on the display screen to signal that an input is needed is called a prompt.

pseudo-random-number—A pseudo-random number is logically produced, but the set of calculations used to generate it are so complex that an outcome cannot be logically deduced by a human being. Computers output pseudo-random numbers as opposed to random numbers, the latter being generated by chance.

RAM—An acronym for random-access memory, RAM is the user programmable internal memory of a computer. The computer can store values in distinct locations in random access memory and recall them again, or alter and restore them. The values in RAM are lost when the power is turned off.

record—A record is a collection of related data elements. When records are combined into relational groups, they are called a file

register—A register is a storage area in memory having a specified storage capacity, such as a bit, a byte, or a computer word, and intended for a special purpose.

reserved word—A reserved word has a predefined meaning in a specific computer language. Reserved words have special meaning and include all BASIC commands, statements, function names, and operator names. Reserved words may not be used as variable names.

RF modulator—An RF modulator accepts an audio or video input and then place this information on a carrier, usually at radio frequencies. Modulators are used to connect computers to television receivers.

ROM—An abbreviation for read-only memory, ROM holds important programs or data that must be available to the computer when first activated. Information in ROM is unalterable and does not disappear when the power is turned off.

scientific notation—A method of expressing numbers that are extremely large or small by using a base number or mantissa multiplied by 10 raised to some power is known as scientific notation. 2,000,000 may be represented in scientific notation by 2E6. The E6 stands for 10 raised to the sixth power, or 1,000,000.

scrolling—When a screen is filled, the display moves upward, or scrolls, one line at a time to allow additional lines to be entered at the bottom. When this occurs, the top line disappears.

software—Software encompasses all types of computer programs, including programs contained in ROM, those stored on magnetic media, and those entered from the keyboard. Any program that can be executed by the computer may be considered as software.

statement—A statement is an instruction preceded by a line number. Some computers allow program lines to contain multiple statements, although TI BASIC does not.

storage—Storage is used to describe a device or medium on or into which data can be entered, held, and retrieved at a later time. Storage may use electrostatic magnetic, acoustic, optical, electronic, or mechanical methods. This term is synonymous with memory.

string—A string is a sequence of items grouped in series according to certain rules.

string constant—A string constant is a sequence of up to 108 characters enclosed in double quotation marks. It is a type of actual value which BASIC uses during execution.

subprogram—In TI BASIC, a subprogram is a general-purpose procedure made accessible by using the call statement. Subprograms are held in ROM and extend the capability of BASIC.

subroutine—A subroutine is a segment of a program that can be executed by a single call. Subroutines are used to perform the same sequence of instructions at different places in a single program.

telecommunications—Telecommunications is data transmission between a computer and remotely located devices.

terminal—A terminal is a part of a computer system that is used for entering or outputting information. It usually includes a keyboard and a monitor.

trace—Trace is a command in BASIC that lets you see the order in which the computer executes statements during a program run. When the trace is in effect, the program line number is displayed as it is executed. This is a valuable debugging aid.

truncation—Truncation is the deletion or omission of a portion of a string. It may also be the termination of a computation process before its final conclusion or natural termination.

update—To update means to modify current information with new information.

variable—Variables are names used to represent values being used in a BASIC program. There are two types of variables: numeric and string. A numeric variable always has a value that is a number. A string variable may only have a character string value and may not be operated on mathematically.

Index

TI-99/4A Game Programs

by Frederick Holtz

Here's an exciting collection of games, puzzles, and brain teasers, all written especially for your TI-99/4A micro! Each one is designed to take advantage of the unique characteristics and capabilities of your machine—graphics, color, *and* sound! And all programs will run with built-in TI BASIC (no Extended BASIC is needed).

Now, at last, you can play games, test your math, spelling, and geography knowledge, challenge a friend to a computer auto race or a game of chance . . . *all without investing in high priced packaged software!* Even if you're a complete computer novice, you can begin playing and enjoying all sorts of games and entertainment programs with step-by-step instructions and line by line descriptions. Plus, the author carefully explains the logic used in creating each game so that you'll be able to modify and improve the games included in this book . . . even get started writing your own original programs.

To give you the understanding you need to fully utilize your micro's capabilities, there's an introduction to your machine's architecture, its use potentials, and special programming characteristics. You'll learn how to use graphics, how to utilize random number generation, how to use arrays, subroutines, loops, and more.

Before you know it, you'll be playing spin the bottle, tic-tac-toe, and demolition derby on your computer . . . playing scrambled word games, card games, roulette, blackjack, and dice . . . testing your skill at number, word, and action games . . . and much more!

If you own a TI-99/4A—or are thinking of purchasing one—you'll definitely want this book!

TAB TAB BOOKS Inc.

Blue Ridge Summit, Pa. 17214

FPT > $11.50 ISBN 0-8306-1630-6

PRICES HIGHER IN CANADA 1095-0883